

"这是当代软件技术领域最重要的著作，
其影响力远远超出编程范畴。"
——Guy Kawasaki
Garage(车库)技术风险投资公司创始人、董事局主席

最新版

大教堂与集市

THE CATHEDRAL & THE BAZAAR

MUSINGS ON LINUX AND OPEN SOURCE BY AN ACCIDENTAL REVOLUTIONARY

ERIC S. RAYMOND 著
卫剑钒 译
BOB YOUNG RED HAT公司董事长兼CEO作序推荐

O'REILLY®



机械工业出版社
China Machine Press

O’Reilly精品图书系列

大教堂与集市

The Cathedral&the Bazaar:Musings on Linux and Open Source by an Accidental Revolutionary

（美）雷蒙德（Eric S.Raymond）　著

卫剑钒　译

ISBN：978-7-111-45247-8

本书纸版由机械工业出版社于2014年出版，电子版由华章分社（北京华章图文信息有限公司）全球范围内制作与发行。

客服热线：+ 86-10-68995265

客服信箱：service@bbbvip.com

官方网址：www.hzmedia.com.cn

新浪微博 @研发书局

腾讯微博 @yanfabook

目录

O’Reilly Media,Inc.介绍

O’Reilly Media通过图书、杂志、在线服务、调查研究和会议等方式传播创新知识。自1978年开始，O’Reilly一直都是前沿发展的见证者和推动者。超级极客们正在开创着未来，而我们关注真正重要的技术趋势——通过放大那些“细微的信号”来刺激社会对新科技的应用。作为技术社区中活跃的参与者，O’Reilly的发展充满了对创新的倡导、创造和发扬光大。

O’Reilly为软件开发人员带来革命性的“动物书”；创建第一个商业网站（GNN）；组织了影响深远的开放源代码峰会，以至于开源软件运动以此命名；创立了Make杂志，从而成为DIY革命的主要先锋；公司一如既往地通过多种形式缔结信息与人的纽带。O’Reilly的会议和峰会集聚了众多超级极客和高瞻远瞩的商业领袖，共同描绘出开创新产业的革命性思想。作为技术人士获取信息的选择，O’Reilly现在还将先锋专家的知识传递给普通的计算机用户。无论是通过书籍出版，在线服务或者面授课程，每一项O’Reilly的产品都反映了公司不可动摇的理念——信息是激发创新的力量。

业界评论

“O’Reilly Radar博客有口皆碑。”

——Wired

“O’Reilly凭借一系列（真希望当初我也想到了）非凡想法建立了数百万美元的业务。”

——Business 2.0

“O’Reilly Conference是聚集关键思想领袖的绝对典范。”

——CRN

“一本O’Reilly的书就代表一个有用、有前途、需要学习的主题。”

——Irish Times

“Tim是位特立独行的商人，他不光放眼于最长远、最广阔的视野并且切实地按照Yogi Berra的建议去做了：‘如果你在路上遇到岔路口，走小路（岔路）。’回顾过去Tim似乎每一次都选择了小路，而且有几次都是一闪即瞬的机会，尽管大路也不错。”

——Linux Journal

译者序

“如果你有正确的态度，有趣的事情自然会找到你”，这是Eric在“大教堂与集市”一文中给出的一条经验。很有意思，2012年3月，机械工业出版社的吴怡编辑通过互联网，在我的博客《“大教堂与集市”读后感》下留了一条评论，问我是否愿意翻译这本书。没有太多犹豫，出于对开源世界的敬仰和（本书所提及的）egoboo的驱动，我决定花费时间和精力翻译这本书。

Eric S.Raymond在开源运动和黑客文化中有着很高的地位，正如他自己所言，他有着开源文化“代言人”、“宣传家”和“大使”等称谓，一个更正式的头衔则是：互联网黑客的观察者/参与者人类学家（observer/participant anthropologist，anthropologist是指“someone who scientifically studies human beings,their customs,beliefs and relationships”）。作为黑客文化的第一理论家，他通过观察和参与，对黑客这个群体的习俗、信念及关系有着深入而广泛的研究。他极力为“黑客”（hacker）一词正名，强调“黑客”并不是媒体报道中的计算机违法分子，而是那种着迷于计算机技术并通过编程提供极具价值软件的人。

软件是人类历史上最奇特的产物之一，它和其他事物截然不同，以至于拿任何事物来比喻软件，都给人以不够贴切的感觉，软件的自动化运行能力和几乎零成本复制能力给人们带来了前所未有的便利，而它带

来的问题（比如版权问题、消费者权益问题、是否开源问题、开发管理模式问题等等）也是多少年来让各界人士争论不休的。

作为软件的核心内容，源代码无疑是软件“王冠上的宝石”，因为软件的核心在于设计，而所有设计都会体现在源码之中，拿到源码，你就几乎拿到了软件的一切。出于对商业利益和市场竞争的考虑，软件制造商本能地希望把源码保护起来，而黑客出于分享、贡献和不重复劳动的考虑，下意识认为开放源码是更道德的事情，他们考虑的是如何更好更快地做事，考虑的是如何创建一个自由自在、为所欲为的软件世界，而不是如何把源码藏起来牟利。

软件设计是一件需要创造力的事情，天才式的软件必然来自天才式的设计，很多优秀软件的最初版本都是由顶尖黑客独自设计和编码的（正如Ken Thompson之于UNIX，Linus Torvalds之于Linux），但软件膨胀到一定程度，由一个人或几个人继续开发维护就不太现实了，对大型软件来说，多人合作似乎是一种必然，但到底多少人合适，如何分工和组织，如何调动程序员的积极性，如何让软件不会因规模和复杂性而失控，从来都有着不同的方法和认识。Eric在本书中向大家展示了两种最为经典且截然不同的模式：大教堂模式和集市模式。传统大型软件公司的开发模式就像是艰难而缓慢的大教堂建造工程，它有着严密的管理和封闭的集中式结构，但在创新上、生产力上和Bug控制上却落后于集市模式。集市模式是一种并行的、对等的扁平化开发结构，其参与者大多来自于互联网上的志愿者，结构松散，来去自由，就像是一个乱糟糟的

集市，但就是这样的组织形式，却取得了像Linux这样令人惊叹的成功。

开源会走向怎样的未来？我们可以看到，互联网和移动智能终端已经日益影响着每个人的日常生活，而你每天访问的互联网网站，绝大部分基于开源的操作系统、Web服务器和数据库，你所使用的智能手机多采用Andriod或iOS系统，Andriod源于Linux，iOS源于开源的Darwin（Darwin则基于开源的Mach和FreeBSD开发），可以说，只要你上网或使用智能手机，你就在不知不觉中使用了开源软件。开源对软件业和互联网带来了巨大影响，并正在和将会对人们的工作和生活产生更显著的影响，正如Eric在前言中所说的，对于任何一个对计算机有所依赖的人，对于任何一个要在未来工作和生活的人，了解一些开源文化，都是很有意义的。

本书的翻译进度超出预期，一方面是我工作本身比较繁忙（正如Eric所言，IT这个行当有什么时候是不忙的），只能在周末不加班时抽出点时间翻译。另一方面，Eric写文章喜欢用很长的句子，并夹杂很多典故和背景知识，如何准确而通畅地表达原文，并不是一件轻松的事。

翻译本着忠实、通达和易懂的原则开展，我尽量忠于原文并使用平实和简洁的语言翻译，如果同一概念对应多个中文词汇，我会选择最常用和最不容易误解的词汇。比如“noospher”有“智域”、“心智层”、“智力圈”和“人类圈”等等译法，考虑到可以和“大气层”相类比而且最容易望文生义，我采用了“心智层”一词；本文出现最多的词汇“hacker”，则当

仁不让地要翻译为“黑客”，这里的“黑客”指的是其本意，即那些着迷于技术并充满才华和理想的人（涉嫌计算机犯罪的cracker则译为“骇客”）；“hackerdom”一词有人译为“黑客道”，但“道”的含义很广泛，容易让人摸不清其内涵，本文将其译作“黑客圈”，因为dom后缀是指具有共同特征或社会地位及角色的群体；“peer”是指在地位上同等的人，如同级别、同类别或同年龄段的人，“同辈”、“同行”、“同类”、“同群”等词汇都不够精确，本文最终选用了虽然不常见但较为精确的“同侪”一词，但将peer-review翻译为“同行评审”，因为软件工程学中已大量使用这一译法。

这本书除了讲述开源及黑客文化外，还给出了很多充满智慧的观点和非常有趣的概念，如命令体系、礼物文化、以少成多、内部市场、竞次、委员会设计、模因、SNAFU现象、软件是服务行业、组织结构决定产品结构、准入门槛越低稳定性越高、程序员是资产而非成本，等等，相信这些内容会给读者带来一些有益的启示和思考。

值得说明的而是，书中很多URL已经失效而无法打开，以后能打开的概率也不大，之所以仍然保留，一方面是为了忠于原著，另一方面，也许你可以从URL中获得一些有用的信息。

最后，感谢我的家人尤其是我妻子于林月的支持与付出。感谢吴怡编辑的耐心和鼓励。感谢译言网上“翻译即反逆”、“nc”、“异议”等人的无私帮助。感谢开源。

谨以此书纪念Robert Anson Heinlein

他教导我：要尊重能力，要珍视和捍卫自由，特别是：昆虫才讲究技能专一。

序

自由不是一个抽象的商业概念。

任何行业的成功几乎都直接和这个行业供应商及客户所享有的自由度相关，对比美国电话业在AT&T失去垄断地位前后的创新步伐，就能知道用户享有选择的自由是多么重要。

计算机硬件行业和软件行业的对比，是体现自由给行业带来益处的最好示例。在计算机硬件行业，供应商和消费者在全球范围内都享有很高的自由度，所以该行业在产品和客户价值方面的创新速度，是人类前所未见的。而在软件行业，其变化则几乎以十年为单位，办公套件是20世纪80年代的杀手应用，其地位直到90年代才受到浏览器和Web服务器的挑战。

开源软件给软件行业带来的自由，可能要比硬件行业制造商和客户所能享受的自由更广阔。

计算机语言之所以被称为语言，是因为它们确实是语言。掌握编程语言技能的社会成员（如程序员）可以使用该语言构建和交流思想，从而做一些有利于其他社会成员（包括其他程序员）的事。在现代社会，人们越来越依赖于软件服务，而对这些软件内在知识的法定获取限制（如软件行业长期以来使用的专有软件 [1]许可制度），导致自由更少、创新更慢。

在软件行业自认为所有基础架构都已经定型了的时候，开放源码这

种革命性的理念横空出世。和以往限制用户获得源代码并以此控制用户的做法不同，开源使用户能够控制他们所使用的技术。将开源工具推向市场需要新的商业模式，相比那些仍然试图控制消费者的公司，开源能给用户带来极为独特的好处，能开发出开源商业模式的公司，将取得非同寻常的成功。

要想对世界做出实质性的改变，开源需要做到这两点：一是要让人们广泛使用开源软件；二是要让用户知道并理解这种软件开发模式能给他们带来的益处。

Eric Raymond把这种革命性软件开发模式的好处解释得如此清晰、透彻、准确，对开源软件革命的成功、Linux操作系统的广泛采用、开源软件供应商及开源用户的成功，都起到了至关重要的作用。

——Bob Young,Red Hat公司董事长及CEO

[1] 专有软件（Proprietary software），又称专属软件、私有软件或封闭性软件等，指通过法律或者技术上的手段，允许用户在一定条件下使用软件，但限制用户对软件进行修改、再发布或逆向工程等，最常见的技术限制方式是封存人们可以读懂的源代码，只发布计算机才能读懂的程序（如二进制格式）。——译者注

前言

——为什么你应该关心这些

你手头这本书是关于黑客[1]行为和文化的，本书收集了一系列文章，最早是写给程序员和技术管理人员的，你可能很自然（并且再正常不过）要问：“我为什么要关心这些？”

最显而易见的答案是：计算机软件在世界经济和商业战略决策中扮演着越来越重要的角色。不管你是由于什么原因打开这本书，大概你对那些关于信息经济、数据时代、互联世界的说法都听得太多了，我不想在这里复述那些。我只是想说，如果我们能对如何编写更优质、更稳定的软件有更深入一点的认识，都将会产生滚雪球般的深远影响。

本书并没有给出基础理论上的创新，只是描述了这样的事实：开源软件系统性地利用开放式开发和分布式同行评审（peer review），不仅降低了成本，还提高了软件质量。开源软件并不是一个新概念（这种文化可以追溯到30年前互联网刚诞生的时候），但直到最近，才在技术和市场的合力下，从小众的圈子中走了出来。今天，开源运动正积极投身于定义21世纪的计算基础设施，任何一个对计算机有所依赖的人，都很有必要去理解它。

我之所以提到“开源运动”，是因为它有着其他更有意思的原因值得读者关注。30多年来，互联网原生的一群充满活力的倡议者团体，一直

在追求、实现并珍爱着开源思想。这些人以自称“黑客”为荣，这里所说的“黑客”并非是记者们滥用的电脑犯罪分子的代称，而是指对某种事物的狂热爱好者、艺术家、古怪的天才发明家、问题解决高手和技术专家。

数十年来，默默无闻的黑客团体不仅要奋力解决技术难题，还要忍受来自于社会主流的冷漠和排斥，直到最近他们才迎来了自己的春天。他们创建了互联网、创建了UNIX、创建了WWW，他们还正在创建Linux和开源软件，经历上世纪90年代中期互联网爆炸式的发展后，人们才终于明白这个世界原本早该好好对待黑客们。

黑客文化及其所取得的成功，对于研究人类动机、工作组织方式、专业主义的未来、公司形态等一些基础性问题，以及这些内容在21世纪信息充裕的后稀缺经济时代中如何变化和演进，都提供了一个很好的研究范例。此外，黑客文化还颇具说服力地预示了人类在适应和重塑经济环境方面将会发生的一些深刻变革，因此，对任何一个要在未来工作和生活的人，懂一些黑客文化都是很有意义的。

这本书是我早前发布在互联网上的一些文章的合集，“黑客圈简史”最早写于1992年，此后多次被更新和修订，其他文章写于1997年2月至1999年5月间，在1999年10月做了一些修改和补充，并在2001年1月本书第二版修订时又做了一次更新，但并没有删掉其中比较技术化的部分或者让文章变得更通俗易懂一些。在我看来，能让读者产生一些疑问和思索，比起让读者感觉厌烦或者被低估智商更有礼貌。如果你在文中遇

到一些特定的技术话题、历史典故或者偏僻的计算机术语缩写，尽管跳过去好了，整本书是在讲一个故事，读到后面的时候，也许你就理解了前面没弄明白的地方。

读者还应了解，这些文章是不断更新的，我周期性地把读者的评论及纠错整理出来并融入其中，当然，我本人对书中的任何错误负责。这本书受益于同行评审（类似于对软件代码的评审）过程，采纳了不胜枚举的意见和建议。这里印刷的并不是最终版本，而更像是一个持续研讨的报告，文中所述黑客文化的很多成员，都积极参与了这些研讨。

最后，我必须要表达一下对很多人以及对一连串机缘巧合的欣喜、惊讶和感谢，也正是这些才导致本书的诞生。

特别感谢那些长期以来的友谊和对本书创作的支持，感谢Linus Torvalds、Larry Augustin、Doc Searls、Tim O'Reilly，很骄傲和你们既是朋友又是同事。最要感谢的是Catherine Raymond，我的挚爱、妻子和最长久的支持者。

我是一名黑客，20年来，作为本书所描述的黑客文化的一分子，我有幸和这个世界上一些最有趣、最杰出的人一起共事，一起解决那些让人着迷的问题，并有幸获得几次珍贵的机会，去做出一些真正创新和有用的东西。有太多的人教给我有价值的东西，教给我黑客技术和其他东西，我无法一一列举他们，在这里谨以本书中的文章作为回馈。

这些文章记录了我在不同阶段的发现和体会，在这个迷人的发现之旅中，我学会了以更新、更深刻的视角来看待我长期以来熟悉的工作。

我一直惊讶的是，这样一个简简单单的随笔，居然在开源软件融入主流世界的过程中起到了持续催化的效果。希望读者能够在这些随笔中捕捉那些令人兴奋的精彩片段，在主流商业及客户迈入这一旅途之际，一同感受那些展现在我们面前的令人赞叹不已的美妙前景。

第二版修订注记

受益于本书第一版的读者，第二版做了一些实质性的补充和修改，大体如下：

多少双眼睛才能驯服复杂性、要命的最后期限，关于分支和伪分支更准确的定义，进化不利条件理论、孔雀、牡鹿和开源开发者动机的关系，开源的经济学动力，信息不对称效应，用开源做竞争武器。在“黑客的反击”一文中的一些预言已经在一年后被验证了，这次又增加了一些新的。在附录中新增了fetchmail项目的成长记录。

[1] 黑客（hacker）一词，原指喜欢通过智力和创造性方法挑战难题的人，尤指那些热衷于计算机技术的编程高手。由于媒体报道中出现的黑客事件往往和计算机犯罪相连，导致人们常常误认为黑客是利用网络入侵他人系统的破坏者。事实上，这些破坏者应该被称为cracker，即“骇客”。——译者注

1.黑客圈[1]简史

本文探索黑客文化的起源，探索那些“真程序员”的史前故事，探索MIT黑客们的光辉岁月，以及早期的ARPAnet是如何孕育了第一个网络部落。本文描绘UNIX早期的辉煌以及最后的停滞，描绘来自芬兰的希望，描绘“最后一个真正的黑客”如何成为新一代开源程序员的领袖，以及Linux和互联网主流如何把黑客文化从公共意识的边缘引领到如今的显赫地位。

1.1　序：真程序员

最初，有一种人叫“真程序员”（Real Programmer）。

他们并不这样自称，也不自称为黑客或者别的什么。据他们中的一位回忆，“真程序员”这个称呼是在上世纪80年代以后才出现的。自1945年以来，计算机技术吸引了世界上最睿智和最有创意的人，从Eckert和Mauchly发明的第一台ENIAC开始，就有一批编程爱好者，并或多或少伴随着一种他们自己能意识到的技术文化，他们编软件和玩软件只是出于乐趣。

“真程序员”通常具备工程学和物理学背景，并常常是业余无线电爱好者。他们穿着白色袜子、涤纶衬衫，打着领带，带着厚厚的眼镜，使用机器语言、汇编语言、FORTRAN或者其他一些已经被人们遗忘了的古老的编程语言。

从二战结束到上世纪70年代初，是批处理操作和俗称“大机”（big iron）的大型机（mainframe）盛行的年代，“真程序员”主宰了计算机世界的技术文化。从这个年代起，一些令人敬慕的黑客文化开始流传，包括种种版本的墨菲定律以及仿德语风格的“Blinkenlights”提示语，后者至今还被张贴在很多机房中。

一些在“真程序员”文化下成长起来的黑客，直到上世纪90年代都仍然很活跃。Cray系列超级计算机的设计者Seymour Cray就是其中的佼佼者，据说他通过机器的前面板开关将他自己设计的整个操作系统以八进

制形式导入他设计的计算机之中，并且没有任何错误，这可真是大师级的神作。

“真程序员”文化和批处理计算（尤其是批处理技术）密切相关，随着交互式计算、大学和网络的兴起，“真程序员”文化逐渐衰落，另一个工程师文化诞生并最终演化成今天的开源黑客文化。

1.2 早期的黑客

我们今天所了解的黑客文化，其起源时间大致可定位于1961年，那年，MIT（麻省理工学院）有了第一台PDP-1。MIT技术模型铁路俱乐部（Tech Model Railroad Club）的信号和动力委员会（Signals and Power Committee）把这台机器当作他们最喜欢的科技玩具，并由此发明了一系列的编程工具、俚语以及直到今天仍然依稀可辨的文化氛围。这些早年轶事可以在Steven Levy写的《黑客》[2]（Hackers）一书中觅得踪迹（Anchor/Doubleday 1984,ISBN 0-385-19195-2）。

“黑客”一词大约就起源于MIT的计算机文化。技术模型铁路俱乐部的黑客们，大都成为MIT人工智能（AI）实验室的核心成员，直到上世纪80年代早期，该实验室在AI领域的研究都一直处于领先地位，1969年后，他们的影响逐渐扩展开来，因为在那年，APRAnet诞生了。

APRAnet是第一个横贯美国大陆的高速计算机网络，它从一个由国防部出资兴建的实验性数字通信系统，逐渐成长为一个连接大学、国防部承包商及研究实验室等数百个节点的大网，使得位于各地的研究者能够以前所未有的速度和灵活性交换信息，这极大地促进了合作交流，推动了科学技术的突飞猛进。

ARPAnet的好处远不止这些，这些电子高速公路把散落全美各地的黑客聚集到一起，构成了产生黑客文化的关键力量。这些黑客原先只是在各自隔离的小团体内发展他们短暂的局部文化，现在，他们发现自己

已经俨然是（或者说他们已经把自己打造成）一个网络部落了。

黑客文化的第一批产物——第一个俚语列表、第一篇讽刺作品、第一次有意识地对黑客道德的讨论——开始在ARPAnet上传播开来，尤其是1973年到1975年间通过网络合作完成的第一版“黑客行话”（Jargon File，http://www.tuxedo.org/jargon）。这本俚语字典成为黑客文化的一个定义性文档，并最终在1983年出版为《黑客字典》（The Hacker's Dictionary）。现在第一版已经停印了，其修订版和增补版是《新黑客字典》（The New Hacker's Dictionary），由MIT出版社于1996年出版（3rd edition，ISBN 0-262-68092-0）。

黑客圈在那些联网的大学中——特别是（虽然不全都是）在计算机科学系中——开始发展壮大，上世纪60年代后期，MIT的AI实验室和LCS实验室首当其冲，斯坦福大学的人工智能实验室（SAIL）和卡内基－梅隆大学（CMU）紧随其后，作为当时最繁荣的计算机科学和AI研究中心，这些实验室吸引了大量的优秀人才，不论在技术上还是文化上，这些人都为黑客文化做出了伟大的贡献。

为了更好理解后面发生的事情，我们需要了解一下计算机自身的发展，因为AI实验室的兴盛和衰落，都是由计算机技术的发展变革而导致的。

从PDP-1时代开始，黑客文化的命运就和DEC（数字设备公司）的PDP小型机系统交织在一起了，DEC率先推出了交互式商业计算和分时操作系统，由于其机器灵活、强大且相对便宜，很多大学都购买了DEC

的小型机。

廉价的分时系统成为黑客文化成长的媒介，在整个ARPAnet的生命周期中，大多数时间都是DEC小型机的天下，其中最重要的是PDP-10，这款发布于1967年的机器，几乎是这之后15年内黑客圈的最爱，TOPS-10（PDP-10的操作系统）和MACRO-10（其汇编语言）至今仍被黑客在一些俚语和传说中充满怀旧地提及。

同样也是使用PDP-10，MIT却有些与众不同，他们完全摒弃了DEC为PDP-10写的软件，而是自己写了一个操作系统，即传说中大名鼎鼎的ITS。

ITS即“不兼容分时系统”（Incompatible Time-sharing System），这充分体现了MIT黑客们的态度，他们就是要走“自己”的路。好在MIT人的智慧配得上这种自负，虽然ITS古灵精怪，时不时出点bug，但却充满了才华横溢的技术创新，并似乎仍然保持着单个分时系统的最长连续使用时间记录。

ITS是用汇编语言写的，其应用大都是用AI语言LISP写的。LISP比当时的任何编程语言都要强大而灵活，事实上，25年过去了，它的设计仍然比如今大多数语言都要好。LISP让使用ITS的黑客能解放出来，以一种与众不同和充满创意的方式思考问题，它是MIT黑客们取得成就的主要因素，并且直到现在仍然是黑客圈最喜欢的编程语言之一。

ITS文化的一些技术产物至今仍然被人们使用，EMACS编辑器可能是其中最广为人知的。ITS的很多传说仍然活在黑客们心中，感兴趣的

话，可以看一下“黑客行话”（http://www.tuxedo.org/jargon）。

SAIL和CMU当然不会自甘落后。在SAIL的PDP-10周围成长起来的黑客们，后来大多成为个人计算机及窗口/图标/鼠标软件交互界面的关键人物。CMU黑客们的工作则引领了专家系统和工业机器人技术的大规模实际应用。

另一个重要的黑客文化节点是XEROX PARC，即著名的Palo Alto研究中心。从上世纪70年代到80年代中期，PARC产生了大量极具突破性的软硬件发明，其数量之多令人震惊。目前被广泛使用的鼠标、窗口和图标式软件交互界面，以及激光打印机和局域网，都是在那里发明的。比起80年代才出现的个人计算机，PARC的D系列机器足足早了十年。然而可悲的是，这些天才的先知们，并没有在他们自己的公司内获得荣耀，以至于有个经典笑话说PARC就是专为别人开发绝妙创意的地方。毫无疑问，PARC对黑客圈的影响是普遍而深远的。

ARPAnet和PDP-10文化在上世纪整个70年代得到了迅猛而多样的发展，电子邮件列表（mailing list）除了促进一些专题兴趣小组（special-interest group）在全美范围内的合作外，也越来越多地应用于社交和休闲领域。DARPA对这种“非授权”行为故意睁一只眼闭一只眼，因为它知道付出这点额外流量，就能吸引一整代的聪明年轻人到计算机领域中来，那真是太划算了。

最广为人知的社交类ARPAnet邮件列表应该算是科幻迷们的SF-LOVERS列表了，时至今日它仍然相当活跃，当然，现在是在ARPAnet

演化而成的互联网上。当时还有其他一些开创性的网上交流方式，后来被一些营利性分时服务商推向商业化，如CompuServe、GEnie和Prodigy（后者现在仍被AOL掌控）。

作为这段历史的描述者，我就是通过早期的ARPAnet和科幻迷圈子，从1977年开始接触黑客文化的，我有幸见证和参与了本文所描述的黑客文化的诸多变迁。

1.3　UNIX的兴起

上溯至ARPAnet还远未普及的1969年，在新泽西郊外，一股新生力量开始成长并不断发展壮大，最终使PDP-10文化变得不再重要。这一年，APRAnet刚刚诞生，而贝尔实验室的黑客Ken Thompson，也正在这年发明了UNIX。

Thompson参与了分时操作系统Multics的开发工作，Multics和ITS有着共同的渊源，它是一个验证一些重要观念的试验床，这些观念的着重点在于如何将操作系统的复杂性隐藏在系统内部，不仅让用户看不到，甚至让大多数程序员都看不到。它使得人们可以更简单地使用Multics（以及为它编程！），可以更多去做那些真正有价值的工作。

当Multics逐渐显露出成为“白象”[3]这种庞大而又无用之物的迹象时，贝尔实验室从这个项目退出了（这个系统后来被Honeywell公司推向市场，但从未获得成功）。出于对Multics环境的怀念，Ken Thompson开始尝试将Multics的一些理念和自己的一些想法融合起来，在一台废置的DEC PDP-7上开发一个新的系统。

贝尔实验室另一名黑客Dennis Ritchie为还处于雏形阶段的UNIX发明了一种新的语言：C语言。和UNIX一样，C被设计为好用、限制少和灵活方便的语言，很快，这些工具在贝尔实验室流行起来了，1971年，Thompson和Ritchie赢得了开发一个内部系统（类似我们现在所说的办公自动化系统）的投标，更大地刺激了UNIX和C的内部传播，而

Thompson和Ritchie的雄心远不止于此。

操作系统在传统上都是用汇编语言精心编写的，目的是充分利用机器的效能。Thompson和Ritchie是最早意识到当时硬件和编译技术都已经好到能让整个操作系统用C语言编写的那批人之一。到1978年，整个UNIX环境已经可以成功地被移植到多种不同型号的机器上了。

这是史无前例和影响巨大的。如果UNIX能够在多种不同型号的机器上提供相同的人机界面和相同的功能，它就能成为一个通用的软件环境。机器更新换代时，用户就可以不再购买那些为新机器而重新编写的软件，黑客们则可以在不同机器上使用相同的工具，而不是每次都去做类似发明轮子和钻燧取火的事。

除了可移植性，UNIX和C还有其他的重要优势，它们都是KISS（Keep It Simple,Stupid）哲学下的产物。程序员可以很容易地在脑海中记忆并掌握整个C语言的逻辑结构（这可不同于之前或之后的大多数语言），而不需要去频繁地查看手册。而UNIX则拥有一系列灵活方便的工具程序，每个工具都被设计为可与其他工具组合运用，以方便地实现特定目的。

UNIX和C的组合，很快被证明适用于极为广泛的计算作业，其中很多完全超出设计者的预期。虽然缺乏正式的支持和推广，它仍然在AT&T内部迅速传播开来。到1980年，它已经扩散蔓延到很多大学和研究机构，而数以千计的黑客们则开始考虑在家里使用它了。

早期UNIX文化中的主力机器是PDP-11及其后代VAX。但由于UNIX

的高可移植性，使得它基本上不用改动就能运行在比整个ARPAnet上范围更广的机器上，没人再用汇编语言了，C语言被迅速移植到了各种机器上。

UNIX甚至有了自己的网络——UUCP：低速、不太可靠但便宜。任意两个UNIX机器可以通过普通电话线路，点对点地交换电子邮件，而且这种功能是系统自带的，不需要额外安装。1980年，第一批Usenet（Usenet最初运行在UUCP上——译者注）站点开始交换广播消息，由此形成了一个巨大的分布式电子公告板，并很快在规模上超过了ARPAnet。围绕Usenet，UNIX站点开始逐渐形成自己的网络部落。

由于一些UNIX站点运行在ARPAnet上，PDP-10文化和UNIX/Usenet文化开始在各自的边缘交汇，但它们并不能和谐相处，PDP-10的黑客们倾向于把UNIX团体看成是一群暴发户，与LISP和ITS具有巴洛克式令人着迷的复杂性相比，UNIX使用的工具看上去原始得可笑，“就像拿着石刀和穿着兽皮！”他们嘟囔道。

除此之外，另外一股势力也开始成长，第一台个人电脑在1975年开始进入市场，苹果公司于1977年成立，技术变革在随后几年以令人难以想象的速度发展，微型计算机的发展势头越来越清晰，并吸引着新一代聪明的年轻人，他们的语言是BASIC，这种语言是如此简陋，以至于PDP-10信徒和UNIX爱好者都认为这简直不值得去蔑视。

1.4 远古时代的终结

1980年发生了很多事，三种文化在边缘互相交叠，却各自形成截然不同的技术体系，ARPAnet/PDP-10文化紧紧围绕着LISP、MACRO、TOPS-10、ITS以及SAIL [4]的发展；UNIX和C主要使用PDP-11、VAX以及慢得让人心烦的电话连接；而一群没有组织的微机爱好者则下决心让普通大众都享受到计算机的威力。

这其中，ITS文化仍占据至尊地位，但是MIT的实验室里已乌云密布，ITS所寄身的PDP-10已经开始过时，实验室随着人工智能的首次商业化尝试而四分五裂，受一些新成立公司的高薪职位吸引，实验室里最优秀的人才正纷纷出走（SAIL和CMU的实验室也一样）。

1983年，ITS文化迎来了致命一击，DEC取消了PDP-10的后续项目“木星计划”，以集中精力研制PDP-11和VAX系列。ITS没有未来了，因为它没有可移植性，而且也没人能把它搬到新机器上，在VAX上运行的Berkeley版UNIX成为最出类拔萃的黑客系统。同时，任何一个有点远见的人都能看到，微型计算机风头正劲，看上去似乎要横扫一切。

就在这个时候，Steven Levy写就了《黑客》一书，其中一个最重要的受访人是Richard M.Stallman（Emacs的发明人），作为MIT AI实验室的标识性人物，他坚决反对将实验室研究成果商业化。

Richard M.Stallman（人们更熟悉他的名字缩写RMS，这也是他常用的登录名）离开实验室，创建了自由软件基金会（Free Software

Foundation），献身于生产高质量的自由软件。Steven Levy称赞他为“最后一个真正的黑客”，幸好没被他说中！

RMS的宏大计划是黑客文化在上世纪80年代遭遇变迁的典型例证——1982年他开始用C语言重新构建整个UNIX的克隆，并免费发布，这就是广为人知的GNU（Gnu's Not UNIX，这是一种递归式的缩写）操作系统，GNU迅速成为黑客活动的焦点，而ITS的精神和传统，作为以UNIX和VAX为主的新一代黑客文化的重要组成部分，籍此得到了保全。

事实上，在其后大约十多年里，RMS的自由软件基金会在很大程度上定义了黑客文化的公共意识形态，Richard M.Stallman本人则毋庸置疑地成为了整个黑客文化部落的唯一精神领袖。

1982到1983年间，集成电路和局域网技术对黑客圈产生了重要影响，以太网和摩托罗拉68000微处理器成为一个很有潜力的组合，一些创业公司纷纷成立，设计开发第一代我们现在所说的工作站。

1982年，一批来自Stanford和Berkeley的UNIX黑客创立了Sun Microsystems公司，他们认为，UNIX操作系统配以相对便宜的基于68000微处理器的硬件，将会被证明是一个可用于多种场合的无敌组合。他们是对的，他们的洞察力为整个产业提供了范例。虽然工作站的价格对大众个体来说还是太贵，但对企业和大学来说已经很便宜了，工作站之间组成的网络（每个用户一台机器），迅速取代了那些过时的VAX机器和其他分时系统。

1.5 “专有UNIX”时代

1984年，当Ma Bell [5]被拆分后，UNIX第一次成为AT&T支持的产品，黑客圈形成了两大阵营，一边是围绕Internet和Usenet而形成的相对有凝聚力的“网络部落”（他们中绝大多数使用运行着UNIX的小型机或工作站级别的机器），一边则是没有网络的分散在各个角落的微机爱好者。

这一阶段，一些严重的计算机破坏（cracking）事件开始被主流媒体报道，记者们误用黑客（hacker）一词来形容那些破坏者，这种不幸的误用一直延续至今。

Sun和其他公司生产的工作站级别计算机给黑客们打开了一片新的天地。其设计目的是实现高性能图形运算和共享数据的网络传输。20世纪80年代期间，黑客圈殚精竭虑，开发了一系列可以充分利用工作站特性的软件和工具。Berkeley UNIX提供了对APRAnet协议的内置支持，解决了由于UUCP点到点连接较慢而带来的网络问题，促进了互联网的进一步发展。

在设法充分利用工作站图形能力的若干尝试中，最流行的当属X Window系统，它由MIT开发，吸纳了十多家公司数百名员工的贡献。X Window成功的关键在于其开发者愿意遵守黑客道德免费提供源码，而且是通过互联网发布。X战胜专有图像处理系统（包括Sun公司自己的）成为了这种改变的一个重要标志，在几年后，它深深地影响了整个

UNIX。

ITS和UNIX之间仍然会时不时爆发一些派系之争，且大多数情况下都是ITS一方挑起的。但随着1990年最后一台ITS机器的关机停用，狂热分子们也不得不放下立场，伴随着不同程度的抱怨，他们大多融入了UNIX文化。

在20世纪80年代已经连上网络的那些黑客群体间，最大的对立来自于Berkeley UNIX和AT&T UNIX的爱好者。偶尔你还可以发现那个时期的招贴画：卡通化的“星球大战”X翼战机从画着AT&T标识的死星的爆炸中疾驶而出。Berkeley黑客们喜欢把自己比做是反抗军，矛头直指那些没有灵魂的商业帝国，而AT&T UNIX虽然在市场份额上从来没有超过BSD/SUN组合，却赢得了标准之战。1990年，AT&T UNIX和BSD UNIX已经很难区分，因为彼此都吸纳了对方的很多新特性。

随着20世纪90年代的到来，已经有十多年发展的工作站技术，受到了明显的威胁，基于Intel 386系列芯片的廉价且高性能的个人计算机出现了，历史上第一次，黑客个人有能力购买一台家用机器，而且其性能和存储能力可以媲美十年前的小型机！UNIX则有能力提供运行于其上的整个开发环境，并能连上互联网。

MS-DOS世界仍然无知并快乐着，早期的微机爱好者们很快扩张成一支庞大的队伍，DOS和Mac黑客们的数量已经超过了“网络部落”，但他们没有产生一种有自我意识的文化，其间有五十多种技术如蜉蝣般生死交替，但从来没有稳定到可以发展出俚语、传说和轶事这类的公共传

统文化。另外，由于一直没有出现类似UUCP或互联网这种真正能流行起来的网络技术，他们也没能发展出自己的网络部落。

CompuServe和GEnie这类商业连线服务此时已经分布很广了，但由于非UNIX操作系统并不会随系统附送开发工具，因此微机爱好者们很少能从网上获取源代码，更不要说发展出合作开发的风气了。

这一阶段黑客圈的主流，有组织或没组织地围绕在互联网周围，他们中的大多数认同了UNIX的技术文化。他们不怎么关心商业服务，只希望有更好的工具和更便利的网络条件，这一时期的厂商们则承诺便宜的32位个人电脑可以让每个人实现这一切。

但软件呢？动辄数千美元的商业UNIX仍然太贵了。20世纪90年代初，很多公司致力于将AT&T或者BSD UNIX移植到PC级别的机器上，但一直不太成功，价格也没怎么降下来，而且最糟糕的是你拿不到可以修改和重新发布的操作系统源代码，传统商业模式是无法满足黑客这种需求的。

自由软件基金会（FSF）也没有做到，RMS许诺已久的HURD——这个给黑客开发的免费UNIX内核，多年来一直停滞不前，直到1996年才拿出一个可用的内核（尽管在1990年，FSF就解决了UNIX类操作系统除内核外几乎所有的难题）。

更糟糕的是，20世纪90年代初，人们清楚地看到，十多年来对专有UNIX的商业化努力显然已经失败了。UNIX曾经承诺的跨平台可移植性，在多个专有UNIX版本的争吵声中看不到任何希望，这些专有UNIX

商家表现得如此沉闷、盲目和没有市场能力，以至于微软凭借Windows操作系统，从他们手中抢走很多市场份额。

1993年初，对黑客文化不怀好意的评论者，也许有足够理由认为UNIX和黑客部落就要玩完了。计算机行业媒体上更是不乏这种人的言论，自20世纪70年代末以来，他们几乎每隔六个月就会旧调重弹，说UNIX马上就要灭亡了。

那些日子里，一种常见的观点看上去好像颇有道理：技术上的个人英雄主义时代已经终结，软件工业和新生的互联网，将会越来越多受微软这类大型企业控制。第一代UNIX黑客看上去已经垂垂老矣（Berkeley计算机科学研究组已经动力尽失，并于1994年失去了资助），多么令人沮丧的时光。

幸运的是，在行业媒体的视野之外，甚至大多数黑客视野之外，一些东西正在慢慢成长，并将在1993年年底到1994年取得令人吃惊的发展，最终，它将引领整个黑客文化进入一个全新的方向，并取得做梦也想象不到的成功。

1.6　早期的自由UNIX

FSF一直未能完成的HURD使得Helsinki大学一名叫Linus Torvalds的学生有了施展才能的空间，1991年，他开始为386机器开发自由UNIX内核，使用的正是FSF提供的软件套件。Linus很快获得了成功并吸引了互联网上的黑客们，他们帮助Linus一同开发Linux：一个全功能的UNIX，源代码完全免费，而且可以再发布。

Linux并不是没有竞争者，1991年，在Linus Torvalds早期尝试的同时，William和Lynne Jolitz正试着把BSD UNIX源代码往386上移植，大多数评论者在比较BSD技术和Linus早期简陋的成果之后，都认为BSD移植将会成为PC上最重要的自由UNIX。

Linux最重要的特点不是技术上的，而是社会学上的。在Linux被开发出来之前，所有人都认为，如果软件复杂到操作系统这样的程度，就必须要有一个精心协作的团队，团队要比较小，而且紧密互动，不管是以前还是现在，这都是很典型的开发模式。商业软件、FSF在20世纪80年代开发的如大教堂般宏伟的自由软件以及从Lynne Jolitz最初的386BSD分裂出来的freeBSD/netBSD/OpenBSD这些项目，都是使用这种模式开发的。

Linux几乎从一开始就发展出一条完全不同的路，其开发更像是仅通过互联网合作的大量志愿者的随意之作。在质量方面，没有严格的标准也没有一个强有力的机构来管理，他们只是执行一个简单得有点幼稚的

策略：每周发布，并在接下来几天内获取数百个用户的反馈。他们创造了一种类似达尔文“物竞天择”的选择机制，被选择对象则是开发者们所做的种种软件修改。让所有人吃惊的是，这种方式工作得非常好。

1993年年底，Linux在稳定性和可靠性上已经和很多商业UNIX不相上下，并能支持比商业UNIX要多得多的软件，一些商业应用软件甚至开始考虑移植到Linux上。Linux间接导致多数小型专有UNIX供应商的关张——因为再没有开发者和黑客买他们的东西了。BSDI（Berkeley系统设计公司）作为少数几个幸存者之一，之所以仍然活跃，是因为他们提供整套的BSD UNIX源代码，并和黑客社团仍保持着紧密的关系。

对于当时的这些新情况，黑客内部并没有做过多评论，外界媒体就更没有什么声音了。黑客文化置那些不断预言自己死亡的言论于不顾，开始以自己的观点重塑商业软件世界，再过五年多，这种趋势会更加明显。

1.7 Web大爆发

和Linux早期发展相互促进的是：公众发现了互联网。20世纪90年代早期，ISP(互联网服务提供商)行业开始逐渐繁荣起来，普通大众一个月花不了多少美元就可以连上互联网，WWW发明以来，互联网本来就很快的增长速度更是加速到了不可思议的地步。

1994年，Berkeley UNIX开发团队正式关闭了，若干种不同版本的自由UNIX（Linux和386BSD的后裔）成为黑客活动的主要焦点。Linux开始被商业公司刻录在CD-ROM上发布，并且非常畅销。1995年年底，主要的计算机公司开始大张旗鼓地宣传他们的软硬件可以很好地连接互联网。

20世纪90年代后期，黑客圈的活动中心是开发Linux和宣扬互联网，WWW使互联网成为大众媒体，许多20世纪80年代和90年代早期的黑客，开始收费或者免费向公众提供ISP服务。

互联网成为主流后，黑客文化开始受到尊敬，并有了一定政治影响力，1994年到1995年间，黑客的大规模强烈抗议，使得试图将“强加密”算法置于美国政府控制之下的Clipper提案无疾而终。1996年，黑客动员起广泛的同盟，导致所谓的“通信合宜法”（CDA）被废止[6]，阻止了政府对互联网的审查。

伴随着CDA的胜利，我们从历史迈进了现在，接下来，我将更多以行动者而非观察者的身份出现（我自己都没想到），更多的故事将

在“黑客的反击”里一一展现。

[1] 黑客圈（hackerdom，或译为“黑客界”）是指整个黑客团体（dom后缀在英文中是指具有共同特征或社会地位及角色的群体）及其文化，有人曾将其译为“黑客道”。——译者注

[2] 《黑客》中文版已由机械工业出版社引进出版，书号是：978-7-111-35840-4。——编辑注

[3] 白象是一种罕有的白色亚洲象，常作为宗教或王室权利的象征，用来形容昂贵、无用且需要很高代价来维持或经营的事物。——译者注

[4] 这里的SAIL指的是SAIL实验室（Stanford Artificial Intelligence Laboratory）推出的SAIL语言（Stanford Artificial Intelligence Language）。——译者注

[5] Ma Bell是Bell System公司的口语化称呼，它以AT&T公司为母公司，下属众多子公司和研究所，由于长期垄断美国电信业，1984年因“反托拉斯法”被拆分。——译者注

[6] 1996年2月，为限制和阻止网上色情内容对青少年的影响和危害，美国总统克林顿签署了《通信合宜法》（Communications Decency Act，CDA），很快，美国公民自由联盟（ACLU）以该法侵害美国宪法第一修正案赋予公民的言论自由权利为由，对美国政府提出起诉。1997年6月26日，美国最高法院终审裁定CDA违背美国宪法，并宣布即刻废止。——译者注

2.大教堂与集市

Linux有一套令人吃惊的软件工程理论，fetchmail作为一个专门对这些理论进行实验的开源项目，所取得的成功让我感到惊讶。我将在本文讨论这些理论，并对比两种完全不同的开发模式：绝大多数商业公司所采用的“大教堂”模式和Linux世界采用的“集市”模式。两种模式的根本不同点在于他们对软件排错有着完全对立的认识。我从Linux的经验出发，证实了这样一个命题：“只要眼睛多，bug容易捉。”这和那些由利己个体组成的自纠错系统有着异曲同工之妙。在本文的最后，我探讨了在这种观念的影响下，软件可能拥有的未来。

2.1　集市模式的成功

Linux是颠覆性的，就在5年前（1991年），谁能想到，几千名散布在全球各地的开发者们，利用业余时间，仅仅是通过Internet这种脆弱的合作，就鬼斧神工般地造就了一个世界级的操作系统？

我肯定想不到。在1993年初Linux进入我视野的时候，我已经在UNIX和开源领域有10年开发经验了。我是20世纪80年代中期GNU最早的贡献者之一，当时我已经在网上发布了一些开源软件，而且还正在开发或者与人合作开发一些程序（如nethack、Emacs的VC和GUD模式、xlife等），这些程序直到现在仍然被广泛使用着，我想我懂这个。

Linux推翻了很多我以为我懂的东西，多年以来，我一直在宣扬“小工具”、“快速原型法”以及“演化式编程”等UNIX信条。但我也相信，如果超过了一定的复杂度，更集中式的管理和更严格的流程是有必要的。我相信大多数重要软件（操作系统和真正大型工具如Emacs编辑器）需要像建造大教堂那样，在与世隔绝的环境下，由天才式专家或几个行家里手精心打造，不成熟时绝不发布beta测试版。

Linus Torvalds的开发风格是：早发布、常发布、委托所有能委托的事、开放到几乎是混乱的程度，这些都令人感到惊讶不已。在Linux社区里，没有建筑大教堂那样的安静和虔诚，倒更像是一个乱糟糟的大集市，充满了各种不同的计划和方法（Linux的文件服务器就是个很好的例子，这里可以接受任何人的代码和文档提交），而既稳定又一致的一

个操作系统就这么诞生了，这真是奇迹中的奇迹。

而事实上，集市模式真的管用，而且非常管用，这让所有人震惊。我开始以自己的方式去了解这种模式，除了在我的个人项目中努力探索外，我也试着去理解为什么Linux世界没有在混乱中四分五裂，反而以大教堂建筑者们难以想象的速度变得越来越强大。

1996年年中，我慢慢开始理解了，而且有幸拥有了一个可以测试我的理论的机会，这个机会使我可以有意识地在集市模式下尝试一个开源项目，我这么做了，更有意义的是，它成功了。

我要讲述的就是这个项目的故事，通过这个故事，我将引出一些在开源开发中很有用的格言警句。虽然对我来说，这些不都是从Linux中学到的，但我们可以看看Linux是怎样淋漓尽致地运用这些理论。如果我是对的，这些格言警句会帮助你准确地理解到底是什么让Linux社区能够源源不断地产生这么多好软件，而且，也许这些格言还能帮助你成为一个富有成效的人。

2.2　邮件必达

“切斯特互联”（Chester County InterLink，CCIL）位于美国宾夕法尼亚州（Pennsylvania）的西切斯特郡（West Chester），是一家小型的免费互联网服务提供商。1993年以来，作为联合创始人，我曾一直负责CCIL的技术部分，并编写了我们独创的多用户论坛程序——你可以通过telnet连到locke.ccil.org来试试看。时至今日，这个程序通过三十多条线路支持近三千名用户访问。这份工作使我能够使用CCIL的56K线路保持每天24小时连在网络上，当然，这是工作需要！

我已经习惯了使用电子邮件，但每过一会儿就telnet到locke上查一下邮件，是一件比较烦人的事。我很希望邮件能够自动地递送到snark（我家里的机器）上，这样，邮件来的时候就会通知我，然后我就能用本地工具来处理它。

互联网原生的邮件转发协议SMTP（Simple Mail Transfer Protocol，简单邮件传输协议）并不适用，因为它最适用于机器一直在线的情况，而我家里的机器并不总是在线，而且也没有一个静态IP地址。我需要这样一个程序，它可以在我时断时续的拨号上网期间，把我的邮件取到本地。我知道有这种软件，它们大多使用了应用层协议POP（Post Office Protocol，邮局协议）。现在绝大多数常见邮件客户端都支持POP，但在那个年代，我所用的邮件阅读器并不支持。

看来我需要一个POP3客户端。于是我便到网上找了一个（事实上我

找到了三四个），使用了一段时间后，我发现它有个明显的缺陷：它不能正确地解析取回邮件的邮箱地址，导致我不能正确回复。

问题是这样的：比如locke上有个叫joe的人给我写信，我把信取到snark上并试图回复时，邮件程序会傻乎乎地想把它发给snark上并不存在的joe。所以我不得不每次把回复邮件地址修改为@ccil.org的样子，这实在有点痛苦。

很明显这种事应该由计算机搞定，但没有任何一个现成的POP客户端能做到这点！这给我们上了第一课：

1.好的软件作品，往往源自于开发者的个人需要。

按说这是显而易见的（正如老话说“需要是发明之母”），但太多的软件开发人员并不需要也不热爱他们正在开发的软件，他们把编程当差事，为的只是拿薪酬。Linux世界里可不是这样——也许这可以解释为什么Linux社区里原创软件的平均质量是如此之高。

那么，我是不是应该立即投入到疯狂的编程中，做出一个全新的POP3客户端和其他程序一较高下？千万不要！我把手头的POP程序看来看去，认真思索“哪个最接近我想要的”，因为：

2.优秀的程序员知道写什么，卓越的程序员知道改写（和重用）什么。

我不敢说自己是卓越的程序员，我只是模仿他们。卓越程序员们有个很重要的特征是“建设性懒惰”，他们知道人们要的是结果而不是勤奋，而从一个部分可行的方案开始，明显要比从零开始容易得多。

以Linus Torvalds为例，他并没有尝试从零开始写Linux，而是以重用Minix（一个用于PC机的迷你型UNIX类操作系统）的代码和理念作为开始，虽然Linux中所有Minix代码最终都被移除或重写，但它在Linux成长初期确实起到了类似脚手架的作用。

基于同样的理念，我试图找到一些代码写得还不错的POP程序，作为我开发的基础。

UNIX世界共享源代码的传统使得代码重用变得很便利（这正是GNU选择UNIX作为基础OS的原因，且不论它对UNIX本身的保留意见），Linux世界则把这个传统发挥到了技术上的极限。相比其他地方，从Linux世界多达数T字节的开放源码中，找到一些他人写的“足够好”的代码要可行得多。

不出意外，连同上次找到的，我一共找到九个备选程序：fetchpop、PopTart、get-mail、gwpop、pimp、pop-perl、popc、popmail和upop。我首先选择了Seung-Hong Oh写的fetchpop。我给程序加上了“邮件头重写”功能，并做了一些其他改进，这些改进被作者接受并在1.9版本中发布。

几周以后，我偶然发现了Carl Harris写的popclient代码，同时带来的是一个难题：尽管fetchpop有一些很好的创意（比如使用后台进程模式），但只能处理POP3协议，代码也略显业余（Seung-Hong Oh在那时是个聪明但缺乏经验的程序员，这两点都看得出来）。Carl Harris的代码要好一些，专业而稳健，但缺乏fetchpop中一些重要而精巧的特性

（包括我写的那部分）。

继续完善fecthpop还是转换到popclient？如果转换，我写的那些代码就可惜了，但同时能换来一个更好的开发基础。

一个更实际的转换动机是popclient对多协议的支持。POP3是最常用的邮件服务器协议，但并不是唯一的。fetchpop以及其他备选程序并不支持诸如POP2、RPOP或AROP这些协议，而我还有点想加上对IMAP（Internet Message Access Protocol，互联网消息访问协议，这是最新设计的、功能最强大的邮局协议，http://www.imap.org）的支持，仅仅是为了好玩。

另外，还有一个理论上的原因让我决定转换，这可是早在Linux之前我就学到的：

3.“计划好扔掉一个吧，迟早你会这么做的。”（Fred Brooks，《人月神话》第11章）

或者可以这么说：在你第一次把问题解决的时候，你往往并不了解这个问题，第二次你才可能知道怎么把事情做好。所以，如果你想做对事情，至少要再做一次。[1]

好吧，我对自己说，改写fetchpop是我的第一次尝试，现在我可以换了。

1996年6月25日，我把自己对popclient所做的第一批补丁发给Carl Harris，发现他基本已经失去了对popclient的兴趣。由于popclient代码有点陈旧，还有些小bug没有解决，很多地方都值得改进，我和Carl Harris

很快达成一致：由我来接手这个程序。

项目在不经意间升级了，我不再是对现有POP客户端做一些小打小闹的补丁工作，而是开始维护整个程序。新的想法不时冒出来，我意识到自己也许可以对程序做些大改动了。

在一个鼓励代码共享的软件文化中，这是一种很自然的项目演化方式。我见证了下面这条：

4.如果你有正确的态度，有趣的事情自然会找到你。

当然，Carl Harris的态度更重要，因为他明白：

5.当你对一个程序不再感兴趣时，你最后的责任就是把它交给一个可以胜任的接棒者。

尽管并没有明确提及，但Carl Harris和我都知道，我们的共同目标是做出最好的解决方案。唯一的问题是我能否证明自己是可靠的，一旦我做到这点，Carl Harris就优雅而利落地把程序交接给我。如果有一天轮到我时，希望我也能交接得这么棒。

2.3　拥有用户的重要性

我就这样继承了popclient，同样重要的是，我继承了popclient的用户群。拥有用户是一件很美好的事，这不仅表明你正在服务于某种需要，表明你做对了某些事，如果发展得当，他们还会成为你的开发合作者。

UNIX另一个传统强项也被Linux发挥到美妙的极致：很多用户本身就是黑客。因为可以拿到源代码，这些黑客能极为有效地缩短排错时间，只要给他们一点点鼓励，他们就会帮你查找问题、给出建议并帮助改善代码，这些比你自己做要快得多得多。

6.把你的用户当成开发合作者对待，如果想让代码质量快速提升并有效排错，这是最省心的途径。

这种做法的效力很容易被低估，事实上，连我们这些在开源世界里的人，都极大低估了这种做法的效力，也就是用户越多就越能有效对抗系统的复杂性，直到Linus Torvalds向我们明白地展示这一点。

我想，Linus最聪明和最有价值的成就其实不是构建出一个Linux内核，而是他发明的这种Linux开发模式。有一次我当面向他表达了这个看法，他笑了，平静地重复了他常说的话：“我基本上是个很懒的人，别人做事，我得名誉。”像狐狸那样懒，或者像Robert Heinlein曾经描绘的一个很有名的角色，太懒以至于无所不能。

回顾以往，GNU Emacs Lisp库和Lisp代码资源库可能要算是Linux这种成功方法的先例，和Emacs用C语言写的核心以及其他GNU工具不

同，Lisp代码池是不断更新的，并且在很大程度上是用户驱动的，大多数新想法和原型在达到最终稳定状态前，都会被重写3到4次，和Linux一样，其频繁的“松耦合”合作都是通过Internet实现的。

说实在的，在fetchmail之前我最成功的作品大概要数Emacs VC（版本控制）模式了，当时我通过邮件和其他三人采用了类似Linux的合作方式。直到今天，我只和其中一人见过面（即Richard Stallman，Emacs的作者，自由软件基金会的创立人，参见http://www.fsf.org）。Emacs的VC模式提供了SCCS、RCS以及后来CVS的前端功能，使得用户可以完成“一键式”的版本控制操作。它是从某人写的一个简略而粗糙的sccs.el演化而来的，之所以能取得成功，是因为和Emacs本身不一样，Emacs的Lisp代码可以迅速地完成“发布/测试/改进”循环。

不只是Emacs，还有其他一些软件产品也使用了两层架构和两级用户群，内核使用大教堂模式开发，工具箱(toolbox)使用集市模式开发，比如数据分析和可视化展现的商业化工具MATLAB就是这样，MATLAB和其他类似产品的用户们发现，创新、酝酿和行动最频繁发生的地方总是在产品的开放部分，而这部分的改进也总是由庞大而多样化的用户群完成。

2.4 早发布，常发布

尽早和尽量频繁发布是Linux开发模式中至关重要的一部分，绝大多数开发者（包括我）都习惯性地认为：除非是很小的项目，这么做有害无益，因为软件的早期版本几乎都是问题版本（buggy version），如果早早发布，恐怕会耗尽用户们的耐心。

这种观念使人们更倾向于支持大教堂开发模式，但如果最重要的目标是给用户提供bug尽量少的软件，为什么你只是每六个月（或者更长间隔时间）才发布一个版本，并且在版本发布的间隔里忙得喘不过气来呢？Emacs用C写的内核是按照这种方式开发的，而Lisp库却不是，因为真正起作用的Lisp代码资源库并不在FSF的控制之下，在那里你可以找到新的和不断发展中的代码版本，而且它们都独立于Emacs的发布周期。[2]

这里面最重要的要数俄亥俄州（Ohio）的Emacs Lisp代码资源库，该库早早就拥有了如今大规模Linux资源库所具备的精神和特性，但当时我们几乎没人认真去想想我们在做什么，或者想想资源库的这种存在，是否意味着FSF大教堂式的开发模式有点问题。1992年左右我做过一次认真的尝试，我从这个代码库中下载了很多代码，并将其正式融入Emacs的官方Lisp库中，很快我遇到了类似政治上的麻烦，尝试非常不成功。

但是一年后，Linux越来越广为人知，它明显与众不同而且要健康得

多。Linus开放式的开发策略简直就是和大教堂模式对着干，Linux的Internet资源库生机勃勃，多个不同版本同时流传，而这完全是由Linux内核那前所未闻的频繁发布所驱动的。

Linus把他的用户当作开发合作者看待，并以一种尽可能最有效的方式：

7.早发布，常发布，倾听用户的反馈。

Linus的创新之处，并不完全在于大量采纳用户反馈并快速发布系统版本（这也是UNIX世界多年来的传统），而更多在于将这种做法强化到一种能和系统复杂度相匹配的强度。我们知道他在早期（1991年左右）发布内核的频率会超过一天一次！这要归功于他在发展合作开发群体方面的努力，Linus比其他任何人都更在意如何利用Internet杠杆促进合作，而且他真的做到了。

他是怎么做到的？我能否加以复制？还是说只有Linus Torvalds这样的天才才能驾驭？

我想不是。Linus无疑是一个顶级黑客，想想有多少人能从零开始建造一个完整的具有产品级质量的操作系统内核？但Linux并没有展现出多少令人赞叹的概念性突破。和Richard Stallman或者James Gosling（NeWS和Java的作者）相比，Linus不是（至少现在还不是）一个富有创造性的设计天才，他更像是一个工程实施上的天才，他具备一种避免bug和防范开发走入死胡同的第六感，而且有一种能发现从A点到B点最省力路径的真本事，事实上，Linux的整个设计，都透露着这种

特质，并反映了Linus那种本质上保守而简洁的设计取向。

所以，如果快速发布和利用互联网杠杆效应不是碰巧而为，而是Linus慧眼发现的最省力路径，那么他最想利用的是什么？什么是他最想从这种开发机制中获取的好处？

这样一问，答案就显而易见了。Linus在持续不断地激励和回报着他的黑客/用户，用自我满足感激励他们，用持续改进（甚至每天都有改进）回报他们。

Linus的直接目标就是将投入排错和开发的“人时”（person-hour）最大化，即便这样做可能导致代码不稳定，或者可能因为一些难以消除的严重bug导致用户群流失，Linus也在所不惜，他相信：

8.如果有足够多的beta测试者[1]和合作开发者，几乎所有问题都会很快显现，然后自然有人会把它解决。

或者说得更通俗一些：“只要眼睛多，bug容易捉。”我把它称为“Linus定律”。

最初我的表达是“每个问题都会有人弄明白”。Linus提出异议，他认为那个弄明白并修复问题的人往往不是也没有必要是那个首先发现问题的人，“有人发现问题，”他说，“另有人搞定问题，我可以公开地说，发现问题更具挑战性。”这个改正很重要。我们会在下一节仔细考查排错过程到底是怎样的，但关键在于，Linux模式下排错的两个部分（发现问题和修复问题）通常都很快。

Linus定律道出了大教堂模式和集市模式最关键的区别：在大教堂建

筑者看来，bug是棘手的、难以发现的、隐藏在深处的，要经过几个人数月的全心投入和仔细检查，才能有点信心说已经剔除了所有错误。而发布间隔越长，倘若等待已久的发布版本并不完美，人们的失望就越发不可避免。

对集市模式而言则完全不同，在上千名合作开发者热切钻研每个新发布版本的情况下，你可以假定bug是浅显易找的，或者至少可以很快变得浅显易找。所以你会频繁发布以获取更多的修正，其副作用是良性的：即便发布中有些小问题，你也不会损失太多。

就是这样，这就足够了。如果Linus定律是错的，那么任何一个像Linux内核这么复杂的系统，经过如此多黑客的改动，在无法预见的不良交互影响以及难以发现的“深度隐藏”bug的重压下，应该已然在某个时刻轰然倒塌了。如果Linus定律是对的，它可以很好解释Linux为什么bug相对较少，且连续运行时间能够超过数月甚至数年。

也许人们不该为这个定律而惊讶，社会学家早在多年前就发现，一群专家（或一群无知的家伙）的平均观点要比一个随机选择的人的观点更有预见性，这就是“德尔菲效应”(Delphi effect)。看来Linus的做法表明这个理论甚至也适用于操作系统排错——“德尔菲效应”可以驯服软件开发的复杂性，甚至是操作系统内核开发这样的复杂性。[3]

Linux对“德尔菲效应”提供支持的一个特别之处在于，给定任何一个Linux软件项目，其成员都是自发选择参与的。早前有读者指出，对Linux做出贡献的人并不是没有规律的，他们是这样的个体：他们有很

大的兴趣使用软件、了解其工作原理、尝试解决遇到的问题并实际给出一个明显合理的解决办法。具备这些特点的人很有可能贡献出有价值的东西。

Linus定律也可被表述为“排错可以并行”，尽管调试人员在排错时需要协调开发人员并与之交流，但调试人员之间并不怎么需要协调，也就是说，增加调试人员并不会带来增加开发人员那样的二次方复杂性和管理成本。

理论上，并行排错会由于重复劳动导致效率损失，但从Linux世界的实践来看，这似乎从来不是一个问题。“早发布、常发布”策略的一个效果就是快速传播反馈回来的修复，从而使重复劳动最小化。[4]

Brooks（《人月神话》的作者）曾经在非正式场合说过：“对于一个被广泛使用的软件，其维护成本通常是开发成本的40%或者更多。令人惊奇的是，这个成本受到用户数的严重影响，用户越多，发现的bug就会越多。”

“用户越多，bug越多”是因为增加用户就会增加程序检验的方式。当用户是合作开发者时，这种效应会被放大，每个着手去发现bug的人，都会有不同的视角，并使用各自略微不同的捕捉方法和分析工具。“德尔菲效应”似乎就是因为这种差异而变得有效。在排错这个特定场合下，差异也使得重复工作倾向于变少。

从开发者角度来看，增加更多的beta测试者似乎并不能减少当前“隐藏最深”bug的复杂度，但这种做法会使bug发现的概率变大：某人的那

套环境正好匹配某个问题，使得那个bug容易被他发现。

为防范严重bug给用户带来的影响，Linus有这么一招：在Linux内核版本号上加以标识（可以从版本号看出系统是否稳定——译者注），潜在用户要么选择上一个被标识为“稳定”的版本，要么冒着有bug的风险使用最新版本以获取新特性。这种策略还没有被Linux黑客们系统性地加以模仿，也许他们以后会这样做。事实上，给用户以选择使得两种版本都更具吸引力。[5]

2.5　多少只眼睛才能驯服复杂性

从宏观上观察到集市模式能极大加速代码排错和演化是一回事，但从微观上，结合开发者和测试者的日常行为，完全理解这是怎样做到的以及为什么会这样，就另是一回事了。本节（在本文首版的三年后写就，采纳了阅读本文首版并对照了他们自身行为的开发者们的观点）我们将仔细审视一下其真正机制。对技术不感兴趣的读者可以直接跳到下一节。

理解这个问题的关键在于要弄清楚这个现象：如果报告bug的用户对源码不关心，则其报告通常不会很有用。对源码不关心的用户，往往报告的都是表面症状，他们把自己的运行环境当成是理所当然的，他们不仅省略了重要的背景数据，而且很少给出重现bug的可靠方法。

这里隐含的问题是开发者和测试者对程序有着不匹配的思维模式，测试者是从外往内看，程序员是从内往外看。对于不开放源码的软件开发，开发者与测试者往往局限于自己的角色，各说各话，都对对方倍感沮丧。

开源开发打破了这种困境，由于大家都有真实的源码，开发者和测试者很容易发展出一个共享的表达模式并进行有效的交流。事实上，一个仅描述外部可见症状的bug报告，和一个直接关联到源码的分析型bug报告，对开发者而言简直是天壤之别。

只要能有一个对出错条件在源码级别上的提示性描述（即便不完

整），大多数bug在大多数时间里就很容易被发现。如果你的beta测试人员中有人指出“在第n行有一个边界问题”，或者仅仅指出“在条件X、Y和Z下，这个变量会溢出”，你扫一眼那部分代码，往往很快就能准确找到出错模式并得出修正办法。

所以，如果beta测试人员和核心开发人员都能意识到源代码的作用，就能极大增强双方沟通和合作的效果。相应地，即便在合作者很多的情况下，核心开发人员的时间也会节省很多。

开源方法之所以能节省开发者的时间，另一个原因是开源项目所常采用的沟通模式，以前我习惯使用“核心开发人员”这个术语，主要想区分一下项目核心人员（通常很少，最常见的是一个人，典型情况是一到三人）和外围人员，外围人员通常由beta测试者和潜在的贡献者组成（通常会达到数百人）。

传统软件开发在组织结构上的根本问题由Brooks定律一语道破：“在一个已经延期的项目上增加人手，只会让项目更加延期。”更为一般地讲，Brooks定律指出，随着开发人员数目的增长，项目复杂度和沟通成本按照人数的平方增加，而工作成果只会呈线性增长。

Brooks定律是建立在经验基础上的，人们发现，bug很容易集中在不同人写的代码的交互接口上，沟通/协调的开销会随开发者间接口数的增加而增多，也就是说，问题规模和开发人员间的沟通路径数相关，即和人数的平方相关（更精确地讲，应该是N(N-1)/2，N代表开发者数目）。

Brooks定律（以及随之而来对开发团队规模的恐惧）建立在这样的假设上：项目的沟通结构是一个完全图[2]，即人人之间都沟通。但在开源项目中，外围开发者实际工作在分散而并行的子任务上，他们之间几乎不交流；代码修改和bug报告都会流向核心团队，只有在那个小的核心团队里才会有Brooks开销。[6]

源码级bug报告非常有用的理由还有很多，但都围绕着这个事实：一个错误可能会导致多种症状，因用户使用方式和环境不同而有不同表现。这种错误往往正是那种复杂而狡猾的bug（比如动态内存管理错误或者非确定中断窗口的产物），这些bug很难重现，也很难通过静态分析找到根源，导致软件中长期存在一些难以解决的问题。

如果测试者能给出一个源码级的bug报告（这个bug可能会引起多种症状），尽管只是一个不确定的试探性描述（比如“看上去第1250行有一个信号处理窗口”或者“你在什么地方把这个buffer清零的”），也会帮助开发人员找到解决那些多症状（症状看上去也许并不相关）问题的关键线索，要知道开发人员往往离代码太近而不能发现问题。这种情况下，要想精确说出是哪个bug导致了哪个外部可见问题，通常很难甚至不可能，但如果经常发布的话，就没有必要去知道了，其他合作者会很快去查看他们发现的bug是否被解决了。很多情况下，人们会关心导致问题消失的源码级bug报告，但很少关心是哪次补丁修复了它。

对于复杂的多症状错误，要想从表面症状追踪到实际bug，往往有多种路径，开发者或测试者追踪时经由何种途径，取决于千差万别的个人

运行环境，并且该环境会随时间产生不确定的变化。实际上，每个开发者和测试者在寻找病状症结的时候，都是在“半随机”（semi-random）的变量集合上对程序状态空间进行采样。越是隐蔽和复杂的bug，就越难从技能上保证采样的对症性。

对于简单和容易重现的bug，重点要放在“半”而非“随机”上，此时，调试技能以及对代码和架构的熟悉程度会大显身手。对于复杂的bug，重点就要放在“随机”上了，这种情况下多人共同追踪bug远比少数几个人循序追踪要有效得多——即便这几个人的平均技能要高很多。

当从表面症状追踪到bug的难度不一且难以预测时，并行纠错的效果会更加显著。一个开发人员会循序选取追踪路径，可能一开始就选了一条困难的路径，当然也可能是容易的路径。而在快速发布模式下，如果多人同时尝试对bug进行追踪，很可能某人立刻就发现了最简单路径，然后在比别人短得多的时间内逮住bug。项目维护人员看到后会发布一个新版本，这样在其他更难路径上追踪该bug的人就可以停下来，以免浪费更多的时间。[7]

2.6 何时名不再符实

在研究了Linus的做法并得出他为什么成功的理论后，我决定在我的新项目（当然没有Linux那么复杂和艰巨）里测试一下这个理论。

但我首先要做的一件事，是大幅度重组和简化popclient，Carl Harris的实现很好使，但表现出了一种不必要的复杂性，这在C程序员中很常见。他把代码作为最重要部分，而将数据结构置于辅助地位。结果就是，代码很漂亮，但数据结构设计得有点随意和潦草（至少从一个LISP老手的标准来看）。

除了提升代码和数据结构的质量之外，我重写的另一个目的是想把它弄成一个我完全理解的东西。如果你对一个程序不能了如指掌，而又要负责他的bug修复，那可就不好玩了。

最初一个月左右，我只是继续遵循Cral Harris的基本设计。对程序所做的第一个重大改变是添加对IMAP的支持，我把协议处理部分重新组织成一个通用的驱动和三个方法表（POP2、POP3和IMAP）。这次改动（以及先前的）再次说明了一个程序员应该铭记在心的一般原则（尤其对于C这种并不天然支持动态类型的语言）：

9.聪明的数据结构配上愚笨的代码，远比反过来要好得多。

Brooks在《人月神话》的第9章里说："让我看你的流程图但不让我看表，我会仍然搞不明白。给我看你的表，一般我就不再需要你的流程图了，表能让人一目了然。"历经30年的术语/文化变迁，这个道理依旧

没变。

这个时候（1996年9月初，即从零开始的六周后），我开始考虑给软件换个名字，毕竟它已不再只是一个POP客户端。但我有点犹豫，因为在设计上还没有做出什么真正新的东西，popclient还需要有点我自己的东西。

当popclient学会如何把取来的邮件转发到SMTP端口后，软件就有了根本的变化，我们一会儿再谈这个。先说说前面我提到的用这个项目验证我所发现的关于Linus成功的理论，（你可能会问）我是如何做的呢？有这么几个办法：

· 我尽早发布并频繁发布（几乎从来没有低于10天一次的频率，在高强度开发阶段会一天一次）。

· 我把每一个因fetchmail联系我的人都加到beta列表（是指beta测试人员邮件列表——译者注）中。

· 每次发布新版本时，我都向beta列表发送朋友对话般的通知，鼓励他们参与。

· 我听取beta测试者们的意见，征求他们关于设计决策的看法，当他们发来补丁和反馈时给他们以热情回应。

这些简单措施立刻收到了回报。从项目一开始，我就收到一些质量高到让程序员们垂涎欲滴的那种bug报告，而且常常还附带了很不错的修复方法；我还收到经过深思熟虑的批评；收到粉丝来信；收到很有智慧的软件特性建议。这一切都表明：

10.如果你把beta测试者当做最珍贵的资源对待，他们就会成为你最珍贵的资源。

衡量fetchmail有多成功的一个有趣指标是beta列表（fetchmail-friends列表）的规模，在本文最新一版时（2000年11月），列表成员达到了287名之多，并且每周还增加2到3名。

实际上，列表成员最多时接近300名，1997年5月底我修订这篇文章时发现，成员开始流失。一些人要求我把他们从邮件列表中去掉，而原因很有趣：他们觉得fetchmail已经足够好了，他们不想再看到关于这个项目的邮件往来！对于一个成熟的集市模式项目，也许这是其正常生命周期的一部分吧。

2.7 popclient变成了fetchmail

这个项目真正的转折点是在Harry Hochheiser发来了他的代码草稿之后，看完这段将邮件转发到客户端机器SMTP端口的代码，我立刻意识到，如果能可靠地实现这个特性，其他的邮件投递模式都可以废弃了。

曾经有好几周，我都在一步步对fetchmail进行微调，我觉得fetchmail的界面设计虽然可用但有点乱，不够优雅，选项琐碎且遍布各处，尤其是那个将已收取邮件导出为邮箱文件或打印到标准输出上的选项让我感到很烦，虽然我也说不出为什么。

（如果你不关心互联网邮件的技术细节，尽可以跳过下面两段。）

当考虑SMTP转发的时候，我发现popclient想做的事太多了。它被设计为既是一个邮件发送代理（MTA）又是一个本地的邮件投递代理（MDA），有了SMTP转发，就应该把MDA去掉而成为一个纯MTA，由它将邮件再发给其他诸如sendmail之类的本地投递工具。

在所有支持TCP/IP的平台上都已缺省保证25号端口打开的情况下，为什么还要折腾复杂的邮件投递代理配置或者去设置邮箱的“加锁并添加”（lock-and-append）选项？而采用SMTP转发带来的另一个特别好处是：取回的邮件看上去就像原发的SMTP邮件一样，这可正是我们想要的。

（返回到非细节层面……）

即便你没有读上面那些技术细节，下面还是给你准备了一些重要经

验。我特意模仿Linus方法后所得到的最大收获是“SMTP转发”概念，是用户给了我这个极妙的想法——我所做的只是去理解其意义。

11.仅次于拥有好主意的是，识别来自用户的好主意，有时后者会更好。

很有趣的是，如果你发自内心地谦逊，并承认你欠别人很多，你将很快发现世界会这样对待你：他们认为是你发明了整个软件，而且你对自己的天赋有着得体的谦虚。我们可以看到这一点在Linus身上体现得有多好！

（当我1997年8月在Perl大会上第一次说出这些的时候，黑客泰斗Larry Wall在我前一排，他以一种宗教复兴般的状态大声喊道：“说下去，说下去，兄弟！”所有的听众都笑了，因为他们知道这一点在Perl发明人Larry Wall身上也适用。）

我以这样的精神将项目运行几周后，开始收到同样的赞扬之词，不仅仅来自于用户，也来自一些对此有所耳闻的人。我把这些邮件收藏了起来，如果什么时候我开始怀疑人生，我就把它们拿出来看看:-)。

接下来是两个更基础的、非政治性的经验，适用于所有类型的设计。

12.通常，那些最有突破性和最有创新力的解决方案来自于你认识到你对问题的基本观念是错的。

我曾尝试解决错误的问题，那时我总是想把popclient做成一个MTA和MDA的结合体，并支持各种各样古怪的本地投递模式。而事实上，

应该彻底重新思考fetchmail的设计，它应该是一个纯粹的MTA，成为Internet邮件常规SMTP对话路径的一个部分。

当你发现自己在开发中碰壁时，当你发现自己苦思冥想也很难做出下一个补丁时，通常你不该问自己是否找到了正确答案，而是该问你是否提出了正确的问题，因为也许问题本身需要被重新定义。

于是，我重新定义了我的问题。显然，正确的做法应该是：（1）将SMTP转发做到通用驱动里面；(2)将它设为默认模式；（3）最后，将所有其他投递模式都扔掉，尤其是投递到文件（deliver-to-file）和投递到标准输出（deliver-to-standard-output）的选项。

对第3步我犹豫了一段时间，主要是害怕影响那些依赖其他投递模式的popclient老用户们。理论上讲，他们可以立刻使用.forward文件（如果使用sendmail——译者注）或者其他非sendmail软件的类似功能来获取同样的效果，但在实际操作中，这个转换可能会让人有点头大。

但当我这样做了以后，好处非常明显，驱动代码中最让人厌烦的部分不见了。配置得到根本上的简化——不再需要低声下气地围绕MDA和用户邮箱打转了，也不再担心底层的OS是否支持文件加锁。

并且，信件丢失的唯一途径也不见了——如果你让程序投递到文件而磁盘满了，信件就会丢失。而使用SMTP转发就不会有这种问题，因为SMTP监听程序只在信件被正常投递或者至少被缓存后，才会返回确认消息。

性能也得到提升了（尽管你不是运行一次就能感觉到的），另一个

不太明显的好处是，用户手册也变得更简洁了。

后来，为了处理一些涉及动态SLIP的棘手问题，我不得不恢复了对“通过用户指定的本地MDA投递”的支持，但这个做起来容易多了。

这件事寓意何在？在不损失效能的前提下，不要犹豫，扔掉那些过时的特性吧。Antoine de Saint-Exupéry [3]（在不写经典儿童读物的时候，他是一名飞行员和飞行器设计师）说过：

13.“设计上的完美不是没有东西可以再加，而是没有东西可以再减。”

当你的代码变得既好又简单，你就知道你做对了，在这个过程中，fetchmail有了自己的特点，和它的前身popclient不再一样了。

现在是时候改名字了。和老的popclient相比，新的设计看上去更像是sendmail的搭档，两个都是MTA，sendmail是先“推”(push)后投递，新的popclient先“拉”(pull)后投递。所以，在开工两个月后，我把它重命名为fetchmail。

这个故事还告诉我们一个更通用的道理，不仅是排错过程可以并行，开发和设计也可以并行（而且能达到让人惊讶的程度）。如果你采用快速迭代开发模式，开发和改进过程就可能成为排错过程的一个特例——修复软件原先在功能或概念上的“疏漏型bug”（bug of omission）。

即便是高层次的设计，如果能有很多合作开发者在你产品的设计空间周围探索，也是很有价值的。设想下一滩雨水是怎么找到下水口的，或者说蚂蚁是怎么发现食物的。探索在本质上是分散行动，并通过一种

可扩展的通信机制来协调整体行为。这很有效，就像Harry Hochheiser和我，一个外围的游走者可能会在你旁边发现宝藏，而你可能有点过于专注而没能发现。

2.8 fetchmail长大了

现在我有了一个简洁和新颖的设计，由于我自己每天都用，我知道代码还不错，并且有了一个不断繁荣的beta列表，我逐渐明白我不再仅仅是做一些微不足道的个人编程，做出来的东西也不再只是对少数几人有用。我现在做的程序，是每一个使用UNIX和SLIP/PPP收发邮件的黑客都需要的。

有了SMTP转发功能，它远远地走在了其他竞争对手的前面，并逐渐成为这个领域内的杀手应用，由于它在同类程序中如此优秀，以至于其他对手不仅被抛弃，而且几乎都被遗忘了。

不过，你一定不要一开始就设定这样的目标和结果。你必须要有一个非常强大的设计创意并完全投入，以至于这个结果就像是不可避免、自然而然和预先设定的。要做到这一点，唯一的途径是你有很多创意——或者有一种采纳别人好主意的工程上的决策力，并将这些创意发展到超越其作者所能想象的地步。

Andrew Tanenbaum写了一个简单的用于IBM PC的原生UNIX，其原意是将其用做教学工具（他称之为Minix）。Linus Torvalds将Minix的概念发展到Andrew可能无法想象的地步——并成长为让人赞叹不已的产物。同样（尽管规模较小），我吸收并努力推进Carl Harris和Harry Hochheiser的创意。我们都不是那种有浪漫原创精神的天才式人物，但是，大多数科学、工程以及软件开发都不是天才完成的，在青史上留名

的往往是黑客。

取得这样的结果真是让人感觉太好了——事实上，这正是黑客们所追求的成功！这意味着我必须要把标准设置得更高一些。现在看来，为了让fetchmail有更好的发展，我不应该只是为了自己的需求而写程序，而应该加入和支持一些他人需要的特性，同时还要让程序保持它的简单性和健壮性。

意识到这点以后，我发现首先要实现的最为重要的特性是对集体邮箱(multidrop)的支持，即有能力从集体邮箱（这里保存着一组用户的邮件）中把邮件取出来，然后将其路由到相应的收件人那里。

我之所以决定增加集体邮箱支持，部分原因是一些用户一直在强烈要求，但更主要的原因是，这可以强迫我以一种完全通用的方式去处理邮件地址，从而借此机会清除掉单邮箱实现中的一些bug，事实上这的确奏效了。弄明白RFC 822（http://info.internet.isi.edu:80/in-notes/rfc/files/rfc822.txt）中描述的邮件地址解析规则着实花了我很长时间，不是因为其中有什么地方很难懂，而是里面牵扯了太多相互关联而又繁琐的细节。

从结果看，对集体邮箱的支持是一个很英明的决定。因为：

14.任何工具都应具备预期内的功能，但一个伟大的工具能给你带来预期外的功能。

fetchmail的集体邮箱模式就有一个用户预期外的功能：它能在客户端（译者注：指连往ISP邮件服务器的终端）上保存地址列表并实现别

名扩展，这意味着用户只要有一台个人电脑和一个ISP账号，就能够维护一个邮件列表，而无需总是读写ISP端的别名文件。

我的beta测试者们要求的另一个重要改动是支持8位MIME（多用途Internet邮件扩展）操作。这很容易做到，因为我已经很小心地保持着第8位的纯洁（也就是说，没有染指ASCII字符集中没用的第8位去让它携带程序信息），并不是我预先想到会有人提这个需求，而是我遵从了另一个规则：

15.写网关类软件时，尽可能不要干扰数据流，而且绝不要扔掉信息，除非接收方强迫你这么做。

如果我没有遵循这条规则，对8位MIME的支持就会变得困难且容易出错。现在，我要做的只是阅读MIME标准（RFC 1652,http://info.internet.isi.edu:80/in-notes/rfc/files/rfc1652.txt），并稍稍增加一些邮件头部生成的逻辑。

一些欧洲用户一直要求我增加一个选项限制每次连接取回的邮件数（以便他们控制昂贵的拨号上网费用）。我抵制了很长时间，而且直到现在对此也不太开心。但如果你是为大家写程序，你就不得不倾听客户意见——虽然他们并不付钱给你。

2.9　从fetchmail学到的其他经验

在回到一般性的软件工程话题前，还有几个fetchmail的特定经验值得思索。对技术细节不关心的读者完全可以跳过这一节。

rc控制文件（fetchmail的配置文件——译者注）的语法包括了一些不会被解析的“噪声”关键字，相比传统配置文件只剩下简洁的“关键字－值”配对，fetchmail这种类似英语的语法使文件更易读。

这个想法来自于某个深夜，当时我注意到rc文件的声明越来越像一个微型指令式语言（这也是为什么我把原先popclient中的“server”关键字换成了“poll”）。

在我看来，如果把一个微型指令式语言弄得像英语的话，会更方便人们使用。虽然我很崇尚“让它像语言那样”的设计哲学（正如Emacs、HTML以及一些数据库引擎），但我并不痴迷于“让它像英语那样”。

传统程序员倾向于喜欢那种非常简洁、紧凑和没有一点冗余的控制语法。这是计算资源昂贵年代的文化遗留，在当时看来，解析过程应尽可能简单和节省资源。英语大约有50%的冗余度，很不适宜作为控制语法的模型。

我并不是因此而不喜欢英语语法，相反，提及它正是为了打破传统观念。有了更便宜的计算资源，简洁就不该成为最终目标。对现如今的计算机语言来说，是否便于人类使用要比是否节省计算资源更重要。然而顾虑是有理由的：一方面，解析过程有着复杂性成本——你当然不想

让它复杂到容易出错和困惑用户的地步；另一方面，试图让语言拥有类似英语的语法，往往使这个所谓的“英语”严重走形，以至于这种对自然语言的表面模仿并不比传统语法让人更易懂一些（你可以在很多所谓“第四代语言”和商业数据库查询语言中看到恶果）。

16.当你的语言还远不是图灵完备（Turing-complete）的时候，语法糖[4]会让你受益良多。

还有一个关于信息隐藏的安全话题。一些fetchmail用户要求我改动软件，使rc文件的密码能够以加密形式保存，这样能防范一个窥探者轻易看到密码。

我没有答应，因为实际上这并不能增强安全性。任何人如果获取了你的rc文件的读权限，都有能力以你的身份运行fetchmail。此外，如果他们想要你的密码，他们能从fetchmail代码中剥离出一个解码器，然后获取密码。

给.fetchmailrc中的密码加密，只会给那些没有认真思考的人一种安全假象，一般而言：

17.系统的安全性只取决于它所拥有的秘密。谨防虚假的秘密。

2.10　集市模式的必要条件

这篇文章的早期读者和评论者一直不断提出这样的问题，即集市模式成功的前提是什么，包括项目领导人应该具备什么资格，项目首次发布并建立合作开发社区的时候，代码应该达到什么状态等。

显然，你不可能从零开始实施集市模式[8]。可以用集市模式测试、排错和完善项目，但以集市模式从零开始一个项目是非常困难的。Linus没有这么试过，我也没有。开发者社区从成立伊始，就需要一个可以运行和测试的东西。

当开始建设社区的时候，你需要拿出一个像样的承诺。程序此时并不需要特别好，它可以简陋、有错、不完整，文档可以少得可怜。但它至少要做到：(a)能运行，(b)让潜在的合作开发者相信，这个软件在可预见的未来，能演变成一个非常棒的东西。

Linux和fetchmail在首次露面时，都有着很强和很吸引人的设计。很多人都已正确意识到这点非常重要，并进而得出结论：项目领导人必须要有高度的设计直觉和聪明才智。

但Linus的设计来自于UNIX，我的最初设计则来自于popclient（虽然后来做了很多改动，而且改动比例要远大于Linux）。所以，一个尝试集市模式的项目领导人或协调人，是否真的需要他本人是一个超群的设计天才，抑或他能借力于其他人的天才设计？

我想，一个协调者是否拥有卓越的原创设计能力，并不是项目成败

的决定性因素，但他是否能识别出别人的优秀创意，则一定是最关键的。

Linux和fetchmail都证实了这一点。Linus虽不是一个让人惊叹的原创设计者（前面说过），但他表现出了能识别优秀设计并将其集成进Linux内核的出色才能。而我也介绍过，fetchmail中最强大的设计（SMTP转发）来自他人。

这篇文章的早期读者曾经很抬举地指出，我之所以低估集市模式的设计原创性，是因为我自身有很多原创，所以视之为理所当然。这可能有点道理，相对于编码或调试而言，设计的确是我的最强项。

但问题是，在软件设计上表现得聪明而有原创性，容易养成一个习惯——在应该保持软件健壮性和简单性的时候，你往往下意识把它弄得既华丽又复杂。我曾经就因为这样的错误把项目搞砸，所以在fetchmail中，我尽量避免再犯同样的错误。

fetchmail项目之所以能成功，相信部分原因是我限制了表现自己聪明的倾向。这（至少）反驳了设计原创性是集市模式项目成功关键的论点。再来看看Linux，如果Linus Torvalds在开发过程中努力尝试在操作系统设计上表现出基础性创新，那做出来的内核还能像现在这样稳定和成功吗？

当然，一定水准的设计和编码能力还是需要的。但我认为，如果一个人真的想要启动一个集市项目，那么他的能力应该已在最低水准以上。开源社区内在的声誉评价机制会给人们施加微妙的压力，使那些不

能胜任项目发展的人，不会去发起一个开发项目。至少到目前为止，这个机制非常有效。

还有一种才能，人们通常不会把它和软件开发联系在一起，但我认为，在集市项目中它和设计才能一样重要——甚至更重要：集市项目的协调人或领导人必须要有很好的人际交往和沟通能力。

这应该是显而易见的，为了建立一个开发社区，你需要吸引人们，让他们对你做的事感兴趣，让他们乐于看到自己的贡献。一些技巧可能有助于实现这些，但远远不是全部，你的人格特征也很重要。

Linus是个好人，人们都喜欢他并愿意帮助他，这（和他的项目成功）不是巧合。我精力充沛、性格外向、乐于社交、有一些脱口秀演员般的说话风格和临场反应，这也不是巧合。为了让集市模式运转，哪怕有一点点的人格魅力，都会对你大有裨益。

2.11　开源软件的社会语境

此言不虚：最好的程序一开始只是作者对自己每天遭遇问题的个人解决方案，程序流传开来则是因为作者遇到的问题成了一大类用户的典型问题。这将我们带回规则1，并以一种可能更有用的方式来重申：

18.想要解决一个有趣的问题，先去找一个让你感兴趣的问题。

Carl Harris和先前的popclient是这样，我和fetchmail也是这样。这个道理已经早为人知，而Linux和fetchmail的发展历史让我们关注到更有趣的一点，那就是下一阶段——由用户和共同开发者们组成庞大而活跃的社区，共同促进软件的进化。

在《人月神话》中，Fred Brooks发现程序员的时间是不可替代的，增加开发者进入一个已经延迟的软件项目，只会让项目更加延迟。像我们前面看到的那样，他指出，项目复杂度和沟通成本与开发人员数目的平方成正比，与此同时，工作完成量只会随人数线性增长。Brooks定律已经被广泛地视为真理，但在本文中我们已经通过多种方式论证了开源软件的开发过程不满足这个定律背后的一些假设——并且从实践上看，如果Brooks定律普适于所有开发项目，Linux是不可能完成的。

作为一种后见之明，Gerald Weinberg在经典之作《程序开发心理学》（The Psychology of Computer Programming）中提出了对Brooks定律的重要修正。在他关于“无私编程”（egoless programming）的讨论中，Weinberg观察到，在某些工作场所，开发人员不将代码看作是自己

的“领土”，而是鼓励别人发现其中的bug和潜在改进点，这些场所中软件改善速度之快，与别处相比是不可同日而语的。（最近，Kent Beck在其“极限编程”（extreme programming）技术中提出的结对编程——两个程序员肩并肩共同完成编程——可以看作是一种效仿。）

也许是Weinberg在用语选择上的问题，导致他的分析未能获得应有的认可——用“无私”来形容互联网上的黑客，这可能会让某些人感到可笑。但我认为他的论点在今天看起来比以往任何时候都更有信服力。

集市模式，运用“无私编程”效果的充足能量，强有力地化解了Brooks定律的影响。Brooks定律背后的原理没有失效，但如果有一个大规模的开发群体和一个低成本的沟通机制，Brooks定律的效果将会被其他非线性因素带来的效果淹没，而后者是大教堂模式下看不到的。这很像是物理学上牛顿理论和爱因斯坦理论的关系——在低能量条件下，老的系统仍然有效，但如果把质量和速度推进到足够大的地步，你就会震惊于核爆炸或Linux。

UNIX历史为我们从Linux中获取经验早已做下了铺垫（为验证其有效性，我专门在一个较小规模的项目上拷贝了Linus的方法[9]），那就是，当编码在本质上仍然是个体行为时，真正了不起的作品来自于对整个社区注意力及脑力的有效利用。一个在封闭项目中只靠自己的开发者，将远远落后于这种开发者：他们知道如何创建一个开放的、有改进能力的环境，在这个环境中，上百人（甚至上千人）反馈并提供设计空间拓展、代码贡献、bug定位以及软件的其他改进。

传统的UNIX世界中，有一些因素阻止了把这种方法推进到极致。一方面是各种许可证(license)的法律限制、商业秘密和市场利益，另一方面（现在看来）是当时的Internet还不够好。

在互联网变得便宜之前，有一些地域性的协作社区，他们在文化上鼓励Weinberg的“无私编程”，开发人员可以很容易地吸引到很多高水平的评论者和合作开发者。如Bell实验室、MIT的AI实验室和LCS实验室、UC Berkeley大学——这些地方都是创意的家园，它们充满传奇并依然强劲有力。

Linux是第一个有意识并成功将整个世界作为其人才库的项目。Linux孕育之时万维网（World Wide Web）刚刚诞生，Linux蹒跚学步之时（1993－1994）ISP产业开始起飞，与此同时，主流对Internet的兴趣也开始爆发，我认为这都不是巧合，Internet普及之后，Linus是学会如何运用新规则的第一人。

廉价的Internet是Linux模式得以发展的必要条件，但我认为它还不足以成为充分条件。另一个非常重要的因素是领导风格的形成和协作机制的建立，这是吸引合作开发者加入项目的关键，也是充分利用互联网媒介作用的关键。

但应该是什么样的领导风格和协作机制呢？它们不应建立在权力关系上——即便可以这样，强制型的领导风格也无法产生我们今天所见的成果。

Weinberg引用了19世纪俄国无政府主义者Pyotr Alexeyvich Kropotkin

所著《一位革命家的回忆》（Memoirs of a Revolutionist）中的一段，很好地诠释了这个问题：

“我成长于一个农奴主家庭，在投入积极生活之时，像那个年代所有年轻人一样，我非常相信命令、指示、斥责、惩罚等行为的必要性。但当我早期不得不管理重要事业并和（自由）人打交道时，在任何错误都会立刻导致严重后果时，我开始感悟到按“命令与纪律原则”行事和按“共识原则”行事之间的重要区别。前者在军队检阅时的作用令人钦佩，但在真实生活中却一文不值，想要达到目标，必须要靠众人的齐心协力。”

“齐心协力”正是Linux这种项目所需要的——对Internet上（可以看成是无政府主义者的天堂）的志愿者们使用“命令原则”是根本行不通的。如果某个黑客想领导一个协作项目，想要项目有效地运作和竞争，他就不得不学会如何施行Kropotkin所提出的“共识原则”，招募和激励有兴趣的成员形成有效社区。他还必须学会如何使用Linus定律。[10]

前面我提到的“德尔菲效应”也许可以解释Linus定律。但我常常注意到，生物学和经济学中的自适应系统是更好的类比，Linux世界的运转，在很多方面像一个自由市场，或者像一个由很多利己个体组成的生态系统，系统中每个个体都追求自身效用的最大化，在其共生的过程中，能够自然建立起一种具备自我纠错能力的秩序，这种秩序比任何集中式规划都要精妙和高效。这里，正是“共识原则”达成的地方。

Linux黑客们致力于最大化的“效用函数”，其目的并不是经典意义上

的经济价值，而是自我满足和黑客声望这些无形的东西。（有人把这种动机称为“利他”，但他们忽视了一个事实，即“利他”本身是“利他者”自我满足的外在表现。）按这种方式运转的志愿者文化其实很常见，除了黑客圈，我还长期参与在科幻迷圈子中，不像黑客，科幻迷们早就清楚认识到“egoboo” [5]（个人在团体中声望的提升）是志愿者活动背后的基本驱动力。

Linus成功地将自己置于项目看门人的地位——大多数开发工作是他人完成的，他不断培养大家对这个项目的兴趣直到它能够自我维持下去，这表现出他对Kropotkin“共识原则”的敏锐领会。用这种“准经济”（quasi-economic）视角来观察Linux世界，有助于我们理解“共识原则”是如何应用的。

可以把Linus方法看成是创造一个有效率的“egoboo”市场——把一个个黑客的利己动机尽可能牢靠地牵系到一个艰巨的任务目标上，而这个目标只有在众人持续的合作之下才能达成。正如我在fetchmail项目上所展示的（虽然项目小了点），Linus方法可以被复制并取得很好效果。也许我做得比他更有意识和更有计划一些。

很多人（尤其是那些在意识形态上不相信自由市场的人）把个人导向为主的利己主义文化看成是碎片化的、本位主义的、浪费资源的、不共享和不友善的。这里仅给出一例，就能很有力地证伪这个认识，那就是Linux文档有着令人震惊的广度、深度和质量。程序员痛恨写文档似乎已经成为一个不争的事实，那为什么Linux黑客们还要写出这么多文

档？很明显，Linux自由的“egoboo”市场比那些有重金投资的商业软件公司，能够产生更有道德、更利他的行为。

fetchmail和Linux核心项目都表明，如果对参与者的“自我”做适当奖赏，一个优秀的开发者或协调者可以利用Internet获取多开发者的好处，而不会让项目陷入混乱不堪。所以对Brooks定律，我有如下的对立意见：

19.如果开发协调者有一个至少像Internet这样好的沟通媒介，并且知道如何不靠强制来领导，那么多人合作必然强于单兵作战。

我认为开源软件的未来会越来越属于那些懂得如何玩转Linus定律的人，属于那些离开大教堂并拥抱集市的人。这不是说个人眼光和远大志向不再重要，相反，我认为冲在开源软件最前沿的人，正是凭借自己的眼光和才华而发起项目，并通过构建有效的志愿者社区将之发扬光大。

可能这不只是开源软件的未来。但没有任何闭源开发者可以发动像Linux社区那样庞大的人才库来解决一个问题，也很少有人能雇得起对fetchmail做出贡献的那200多人（1999：600，2000：800）！

可能最终导致开源软件取得胜利的，不是因为“合作是道德正确的”或“软件闭锁 [6]是道德错误的”（也许你相信后者，但Linus和我不这样认为），而仅仅是由于闭源世界不能赢得一场与开源社区之间的不断演化的军备竞赛，因为后者可以在一个问题上投入比前者多几个数量级的熟练技术工时。

2.12 管理与马其诺防线[7]

1997年首版的“大教堂与集市”文章结束于上面这个展望：由程序员和无政府主义者组成的快乐的网络部落，战胜和压倒了等级森严的传统闭源软件世界。

然而，很多怀疑者并不信服，应该对他们提出的问题给予公允的回应。大多数对集市模式的异议都归结到这一点：集市模式支持者低估了传统管理方式带来的生产率乘数效应。

抱有传统观念的软件开发管理者经常会反驳说，开源世界中项目团队形成、变动和解散的随意性，极大抵消了开源社区在人数上相对闭源单人开发者的明显优势。他们说，软件开发真正需要的是持久的努力和客户预期在产品上持续投资的程度，而不是到底有多少人在锅中扔入一块骨头然后让它慢慢炖着。

这并不是没有道理，事实上，我在“魔法锅”（The Magic Cauldron）一文（本书后面的一章）中提出了这样的观点：未来软件产业的经济关键是服务价值。

但这个论点有一个重要的潜在问题，它暗藏了这样的假设：开源开发不能提供“持续的努力”。事实上，有一些开源项目长期保持着连贯的方向和有效的维护社区，而没有任何传统管理认为不可或缺的奖励机制或管控机构。GNU Emacs编辑器是一个极端的、有启发性的例子，尽管该项目的人员流动率很高，从始至今持续起作用的只有一人（Emacs的

作者），但15年来，仍然有数百名贡献者投入努力，他们合作造就了一个统一的架构体系，还没有哪个闭源编辑器能匹敌这样的长寿记录。

这倒是给了我们一个质疑传统开发管理有什么优势的理由（与大教堂和集市模式的争议无关）。GNU Emacs能够在15年内保持一致的架构体系，像Linux这样的操作系统在硬件和平台技术不断变化的8年来亦复如是，很多设计优秀的开源项目发展都超过了5年（事实上确实如此）——我们当然有权利质疑，传统开发管理的巨大花费，究竟给我们带来了什么。

不管是什么，它肯定不是这三项目标的可信履行：最后期限、预算和需求书中的所有功能。能达到其中一个目标，就已经是很少见的管得不错的项目，更不用说三个全达到了。它也不是在项目生命周期内适应科技和经济变化的能力，这方面，开源社区已被证明远远更为有效（这很容易被核实，比如说，比较Internet至今30年的历史和专有网络技术很短的半衰期；或者比较微软将Windows从16位过渡到32位的成本以及同时期Linux完成同样升级的毫不费力——不仅仅是在Intel产品线上，而且包括64位Alpha处理器在内的十多类硬件平台）。

一些人认为购买传统模式产品会带来这样的保障：如果项目出错，有人会负责，并为可能的损失买单。但这只是一个幻觉，大多数软件许可证连对商品的保证都没有，更不用说履行责任了——因软件质量差而成功获得赔偿的案例几乎没有。即便很常见，也不要因为可以起诉某人就觉得心安，你想要的不是官司，而是能用的软件。

那么所有这些管理开销能带来什么？

为弄明白这点，我们需要了解软件开发管理者是如何看待自己工作的，我有位朋友看上去在这方面做得很好，她说软件管理有五个功能：

· 明确目标并让大家朝同一个方向努力。

· 监督并确保关键细节不被遗漏。

· 激励人们去做那些乏味但必要的“体力活”。

· 组织人员部署并获得最佳生产力。

· 调配项目所需的资源。

显然所有这些目标都是有价值的，但在开源模式及其所在的社会语境中，人们会惊奇地发现这些目标毫无意义，我们按颠倒过来的顺序分析。

朋友告诉我，很多资源调配基本上是防守性的；一旦你拥有人、机器和办公空间，你就不得不防备同级管理人员对资源的竞争，以及上级对有限资源中最有效部分的调用。

但开源开发者是志愿者，是因为兴趣和能力（能否对项目有所贡献）自主选择的（即便他们因开源工作领取薪水，这也依然适用），志愿者精神倾向于自发去关心资源问题的“解决”，他们会把自己的资源带到工作中，这里几乎没有传统意义上“防守”的必要。

无论如何，在一个廉价PC和快速Internet连接的世界里，我们发现始终如一真正有限的资源是技术人员的关注，开源项目如果失败了，根本不会是因为机器、网络或办公场地，它们死掉的唯一原因就是开发者们

不再感兴趣了。

这种情况下，开源黑客通过“自我组织”来最大化生产力就显得加倍重要，自愿者自主选择项目，社会环境则无情地选择能力。我那个朋友对开源世界和大型封闭项目都比较熟悉，她相信开源之所以成功，部分原因是开源文化只接受编程人员中那最有才华的5%。她将自己的大部分时间都花在了组织部署其他的95%，并因而第一手见证到那广为人知的差异：最有才华的程序员和那些刚刚及格的程序员之间，生产率能相差100倍。

人们常常会因为这种差异提出一个棘手的问题：对单个项目或者整个行业来说，撤离出那50%能力较差的程序员会不会更好一些？思考已久的管理者们早就明白，如果传统软件管理的目的仅仅是把能力最差那部分人的净损耗转变成微弱盈余的话，那问题就简单多了。

开源社区的成功使得这个问题更加尖锐，强有力的证据表明，从互联网上招募自主选择的志愿者，通常更便宜、更有效。而传统方式管理的那些写字楼里的人，其中有很多可能宁愿去干点别的。

这很直接把我们带到“动机”问题上，一个经常听到的说法和我朋友的观点等效：传统开发管理是对缺乏激励的程序员们的必要补充，否则他们不会干得很好。

这个答案常常伴随着一种观点，认为开源社区只会去做那些很吸引人或者技术上很好玩的东西，其他“无聊”部分则会被丢在一边（或半途而废），等那些受金钱激励的坐在隔间里的苦工们在管理者的严厉鞭策

下机械地将其生产出来。我将从心理学和社会学角度，在“开垦心智层”（Homesteading the Noosphere）一文中对这种观点进行质疑。

如果传统、闭源、严格管理模式的软件开发真的想靠这种由“无聊”部分组成的马其诺防线来防御，那么它之所以在某个应用领域能继续生存下去，只是因为还没人发现这些问题是真正有趣的，并且还没人发现迂回包抄的路径。一旦有开源力量介入这些领域，用户就会发现终于有人是因为问题自身的魅力而去解决它的，就像其他所有需要创造力的工作，若论激励效果，问题自身的魅力比单纯的金钱要有效得多。

单纯为解决动机问题而设立一个传统的管理架构，可能战术不错，但其战略是错误的，这一套在短期看会有效，但长远来说一定会失败。

目前来看，传统的开发管理相对于开源，至少在两方面（资源调配和组织）都没有胜算，而且似乎在第三点（动机）上也快要玩完了。可怜的传统管理者，现在是四面楚歌，在“监管”这个话题上也无法获取一丝安慰，开源社区最强大的一个长项就是非中心化的同行评审，所有致力于细节不被疏漏的传统方法，都无法和它相比。

那么，“明确目标”的作用是否可以作为传统软件项目管理值得存在的正当理由？也许是，但是，比起开源世界由项目领导人和部落长老决定目标的方式，我们需要有好的理由来相信管理委员会和公司路线图所定义的目标在价值上以及在让成员广泛接受上能做得更成功。

但我们很难得出这个结论。并不是因为开源这边（Emacs经久不衰，Linus Torvalds号召大量开发者们“统领世界”[8]）表现太优秀，而是传统

机制在定义项目目标方面做得实在太糟。

软件工程中最广为人知的一条大众定理是：传统软件项目中的60%到70%，要么是从未被完成，要么被他们的用户拒绝。如果这个比例还算靠谱的话（我还没见过任何一个有经验的项目管理者对此提出过异议），那么大多数项目把目标设定得要么太不现实，要么完全错误。

这正是如今软件工程领域中，一听到所谓“管委会”就让人后背发凉的原因——即便（或尤其）听者是一名管理者。以前仅仅是程序员抱怨这种模式，现在，呆伯特[9]玩偶已经出现在主管的办公桌上。

所以，我们对传统软件开发管理者的答复就很简单了——如果开源社区真的低估了传统管理的价值，那为什么你们中的这么多人都表现出对你们自己流程的不屑？

又一次，开源社区的例子把这个问题变得更为尖锐——我们做这些是为了乐趣。我们创造性的游戏已经在技术上、市场占有率上、观念认同上以令人震惊的速度获得了增长，我们不仅证明了我们可以做出更好的软件，而且证明了快乐也是一种资产。

本文第一版发布已经两年半了，我能提供用来作为结束语的最激进的观点，已经不再是“开源统领软件世界”这样的愿景了，毕竟那些很严肃的西装革履的人，也认为这种观点有些道理了。

更进一步，我想给出一个更普遍的关于软件的经验（可能适用于所有创造性或专业性工作），人类通常会从一种位于“最佳挑战区”的任务中获得乐趣，也即它不是太容易而让人无聊，也不是太困难而无法完

成。一个快乐的程序员是一个既没有被浪费也没有被压垮（由于不适当的目标或过程中充满压力与冲突）的人，乐趣预示着效率。

如果你在工作过程中感到恐惧和厌恶（即便你以自嘲的形式来表达——比如悬挂呆伯特玩偶），就应该意识到过程已经出了问题。快乐、幽默和玩兴是真正的资产，前面我之所以写“快乐部落”（happy horde）并不是为了首字母押韵，而用一只憨态可掬的企鹅作为Linux吉祥物也绝不仅仅是为了搞笑。

现在看来，开源成功的一个最重要成果，就是告诉我们，“玩”是创造性活动中最具经济效能的工作模式。

2.13　后记：网景拥抱“集市模式”

感觉自己正在帮助创造历史的感觉实在很奇妙……

1998年1月22日，大约在我首次发布“大教堂与集市”一文七个月后，网景通信公司宣布了开放“网景通信家” [10]源代码的计划（http://www.netscape.com/newsref/pr/newsrelease558.html）。此前，我一点消息也不知道。

很快，网景的执行副总裁和首席技术官Eric Hahn给我发了一封电子邮件：“我代表网景公司所有员工，感谢您帮助我们走到这一步，您的思考和写作，对我们的决定有着至关重要的启发意义。”

接下来的一周，我受网景之邀飞抵硅谷，和他们的高管层以及技术人员开了一个长达一整天的战略会议（1998年2月4日），我们在会议上设计了网景源码的发布策略及许可声明。

几天后我写道：

“网景准备在商业世界中给我们提供一个大规模的、真实的集市模式的测试。开源文化现在面临一个危险：如果网景此举失败，那么开源概念将受到严重怀疑，商业世界将在未来十年中都不会再碰它。

另一方面，这也是一个绝好的机会，华尔街以及其他一些机构，对这件事的初步反应是审慎乐观的。我们也获得了一个证明自己的机会，如果网景通过此举能够重新收复市场份额，那将会引发一场早该到来的软件产业革命。

接下来的一年将会非常有启发性，也会非常有趣。”

事实上确实如此。2000年年中我修订此文的时候，项目（后来被命名为Mozilla）发展的情况只能说勉强成功，它达到了网景的最初目标——阻止微软在浏览器市场上的垄断锁定。当然它也取得了一些引人注目的成就（特别是下一代浏览器内核Gecko [11]的发布）。

然而，它还没能像Mozilla创立者最初希望的那样，聚拢大规模的来自网景外部的开发力量。问题似乎是，在很长一段时间内，Mozilla的发布实际上违反了集市模式的一条基本规则：它的发布缺少能让潜在贡献者可以轻易运行和查看效果的东西（发布后的一年多时间内，编译Mozilla源代码都需要Motif私有库的许可证）。

最消极的是（从外界来看），Mozilla组织在项目开始两年半内都没能发布一款商品级的浏览器。而且在1999年，一位项目核心成员的离职引发了不小的影响，他抱怨项目糟糕的管理和机会的错失，“开源，”他很正确地评价道，“并不能点石成金。”

确实不能。现在（2000年11月）看来，比起Jamie Zawinski写离职信的那个时候，Mozilla的前景有了戏剧性的改善——最近几周的每夜版（nightly releases）已经终于在产品可用性上通过了关键性门槛。但Jamie没有说错，对于一个已有项目，走向开源不一定就能让项目免受目标错误、代码杂乱以及任何其他软件工程慢性病的困扰。Mozilla已经同时示例了如何让开源成功和如何让开源失败。

与此同时，开源理念在其他很多地方已经获得了成功和支持。自网

景发布源码以来，我们看到人们对开源模式的兴趣有爆发式的增长——这种趋势由Linux操作系统推动，并推动Linux继续获得成功，Mozilla引发的潮流将继续加速前行。

[1] beta测试即“β测试”，和“α测试”相对应。对于一个即将面世的软件产品，α测试是指软件公司组织内部测试人员模拟各类用户行为对产品(此时为α版本)进行测试。随后的β测试是指软件公司组织各类典型用户在日常工作中实际使用（此时产品为β版本），并要求用户报告错误及异常情况。最后软件公司再对β版本进行改错和完善。——译者注

[2] 完全图是一个有n个顶点的图，每两个顶点之间都有且只有一条边，也即共有n(n-1)/2条边。——译者注

[3] Antoine de Saint-Exupéry（1900年6月－1944年7月），法国作家、诗人和飞行员，是经典小说《小王子》的作者。——译者序

[4] 语法糖（Syntactic Sugar，或译为语法糖衣）是英国计算机科学家Peter J.Landin发明的术语，是指为计算机语言添加某种不会影响语言功能的成分，但却使其更易用一些，目的是增强代码可读性，避免出错的机会。——译者注

[5] “egoboo”是ego boosting的口语化简称，是指参与志愿工作得到公共认可而获得的快乐，这个术语大约出现在1947年，最早用于科幻迷圈子。egoboo原本是描述人们看到自己名字出现在出版物上的感觉，由于做到这点比较可行的方式是做一些值得被别人提及的事情，该概念很

快用到了志愿者活动中。——译者注

[6] 软件闭锁（software hoarding）是指某人或公司试图阻止别人使用或共享他们的软件源码。——译者注

[7] 第一次世界大战后，为防范德军入侵，法国陆军部长马其诺提议沿法国和德国边境建设一条叫做马其诺防线的防御工事（1929年开始建造，1940年基本建成，长达390公里）。马其诺防线曾被认为是牢不可破的，但德军从侧面包抄并随后占领了这条防线。马其诺防线用来形容表面看似坚固而实际没有价值的东西。——译者注

[8] Linus Torvalds于1999年在“The Linux Edge”一文中提到：“Linux现在有数百万用户、数千名开发者和正在增长的市场，Linux用在嵌入式系统中，用在机器人系统中，用在航天飞机上，我想说我早知道这些都会发生，这些都是统领世界计划的一部分”。——译者注

[9] 呆伯特（Dilbert）是美国作家和漫画家Scott Adams创造的广为流行的职场卡通人物。他是一名计算机工程师，热爱科技、待人友善、憨厚老实，但很不成功，在公司里人微言轻，常常被主管过分要求和利用。——译者注

[10] 网景通信家（Netscape Communicator）是网景通信公司互联网套件（在版本4.0到4.8时）的总称，内含网景导航者（Netscape Navigator）浏览器、电子邮件客户端、新闻组软件、HTML编辑器、多用户通信客户端以及地址簿等辅助工具，在此之前，网景导航者既是整个套件的名字，也是浏览器的名字，容易引起混淆。

[11] Gecko浏览器内核（Rending Engine，又译为排版引擎、渲染引擎）是Mozilla火狐浏览器（FireFox）采用的内核，由于Gecko代码完全公开，因此受到许多人的青睐，使用Gecko内核的浏览器也很多。——译者注

3.开垦心智层[1]

观察到开源软件“官方”意识形态（通过版权许可来定义）和黑客的实际行为存在矛盾之后，我研究了开源世界里处理所有权和控制权的习惯做法，展示了其中隐含的与Lockean土地权理论相类似的财产权理论。然后我将这些习惯关联至黑客的“礼物文化”，也即通过付出时间、精力和创意，参与者在竞争中获取声望的文化。最后，我对黑客文化中冲突解决方法进行了分析和研究，并得出了一些约定俗成的推论。

3.1 关于“矛盾”

任何人只要去观察一下Internet中繁忙和多产的开源软件世界，一定都会注意到一个很有意思的矛盾：他们实际做的和他们所声称的并不一致，也就是说，开源文化的“官方”意识形态和他们的实践并不一致。

文化是有适应能力的机器，开源文化会对一系列可识别的动力和压力做出反应。通常，文化对其所处环境的适应，既反映了其自觉的意识形态，也反映了其隐藏的、无意识或潜意识的认识。而无意识调适和其意识形态的不一致，是很常见的现象。

本文将挖掘这种矛盾的根源，并探明隐藏在其后的动力和压力。我将推导黑客文化和其习惯中一些有趣的东西，并以建议的形式给出结论，以更好发挥黑客文化中这些潜在认识的作用。

3.2 黑客意识形态的多样性

Internet开源文化的意识形态（即黑客们所称的信仰）本身是一个相当复杂的话题。所有成员都认为开源（也即软件可以自由发布、可以迅捷地演化、可以根据需要修改）是一个好东西，值得投入大量的和群体的努力。这种认同很有效地定义了文化中的成员，但是，若论为何形成这种文化，不同个体和开源子文化给出的原因相差甚远。

差异可以体现在热忱度上：开源仅仅是一个便利的手段（好工具、有趣的玩具、有意思的游戏），还是说其本身就是目的？

极度热忱的人可能会说："自由软件是我的生命！我活着就是为了创造有用的、优美的程序和信息资源，并把它们贡献给社会。"中热忱度的人可能会说："开源是件好事，我愿意花大量时间帮助它成功。"低热忱度的人则可能说："开源有时候还不错，我也玩这个，我尊敬那些创造它的人。"

差异还体现在敌对性上：反对商业软件，以及反对那些试图支配商业软件市场的公司。

高度反商业化的人可能会说："商业软件是偷窃和闭锁，我写自由软件就是要终结这种邪恶。"中度反商业化的人可能会说："总的来说，商业软件其实还好啦，毕竟程序员也是要赚钱的，但是一些软件公司用低劣软件充斥市场、滥用权势压制别人，就比较可恶了。"不反对商业化的人可能会说："商业软件没什么不好，我使用或者编写开源软件只是

因为我更喜欢它。”（自本文第一版发布以来，由于软件业开源的部分越来越多，人们可能还会听到“只要我能得到源码，或者它能够如我所愿，那商业软件就是好的。”）

上面的分类能够组合出九种态度，在开源文化中都有所体现。做这种区分是有价值的，因为这暗示着开源文化中不同的目标、适应性和合作行为。

从历史上看，黑客文化中最引人注目和最有组织的部分，都是既狂热又反商业化的，Richard M.Stallman（RMS）创立的自由软件基金会，从20世纪80年代早期开始，支持了大量的开源项目，包括Emacs和GCC——它们至今仍然是互联网开源世界的基础，而且看起来将继续保持这个地位。

多年来，FSF是开源文化唯一的最为重要的焦点，它产生的大量工具，仍然对开源文化至关重要。FSF也是长期以来唯一以机构身份（黑客文化外部可观察到的）对开源进行赞助的组织。他们有效定义了“自由软件”概念，并有意赋予其对抗意味（后来出现的“开放源码”叫法则有意避免这点）。

所以，不管是从黑客文化内部还是外部，人们都倾向于将这种文化标记上FSF那种狂热态度和反商业目标的烙印。RMS本人否认自己是反商业化的，但他的计划被绝大多数人如此解读（包括很多他的忠实信徒）。FSF以生机勃勃和清晰明确的行动致力于“消灭软件闭锁!”，这成为最接近黑客意识形态的行为，而RMS则成为最接近黑客文化首领的人

物。

FSF的许可条文“通用公共许可证”（GPL）表达了FSF的态度，并在开源世界中得到广泛使用。北卡罗来纳的Metalab（http://metalab.unc.edu/pub/Linux/welcome.html，前身为Sunsite）是Linux世界最大和最流行的软件库，1997年7月，大约有一半的Sunsite软件包使用GPL作为许可声明。

在黑客文化中，FSF并不是独此一家，还有很多较为安静、冲突较少和对市场更友好的流派。实用派并不是那么热衷于意识形态，他们更看重一些工程师传统，这些传统来自于开源早期的实践（在FSF之前），其中最重要的部分，来自于UNIX和早期非商业互联网水乳交融而产生的技术文化。

对典型的实用派而言，其反商业化的程度只能算是中等，他们对商业世界的主要抱怨不是“闭锁”，而是商业世界一直拒绝接纳UNIX、开放标准和开源软件这些更好的做法。如果实用派痛恨什么的话，应该不太会是普通意义上的“闭锁”，而是现在软件王国里那似是而非的国王——以前是IBM，现在是微软。

对于实用派来说，GPL更像是工具，而不是行动目标。它的主要价值不是用作武器来对抗“闭锁”，而是鼓励软件分享和集市开发社区的成长。实用派更看重它是否有好的工具和玩意儿，而不太在乎是不是反商业，他们会使用高质量的商业软件而不会产生价值观上的排斥。与此同时，实用派的开源经验教会他们什么是软件的技术质量标准，这个标准

是绝大多数闭源软件都难以达到的。

多年以来，作为黑客文化的一部分，实用派对完全接纳GPL或FSF计划表达了顽固的拒绝。这种态度贯穿20世纪80年代和90年代早期，并往往和Berkeley UNIX粉丝、BSD许可证使用者、众多开源UNIX系统（以BSD源码为基础）的早期发展联系在一起。但这些活动并没有构建起一个稍具规模的集市社区，而是走向了严重的碎片化和低效能。

直到1993至1994年，由于Linux的爆发，实用派才找到一个真正有力的基础，尽管Linus Torvalds从来没有反对过RMS，但他树立了一个友好对待商业Linux成长的榜样，他公开支持在特殊用途上使用高质量的商业软件，并对黑客文化中的纯粹派和狂热人士予以适度的调侃。

Linux快速成长的额外好处是吸引了一大批新黑客，他们对Linux忠心耿耿，而把FSF计划视为过气的兴趣，尽管这一波黑客将Linux系统称为“GNU一代的选择”，他们中的大多数更愿意效仿Torvalds而不是Stallman。

完全反商业化的纯粹派，越来越发觉自己成了少数派。1998年2月网景公司声明开放“Navigator 5.0”源码之时，态势就更为明朗了，企业界对“自由软件”产生了更多的兴趣，而黑客文化则抓住这个空前机会，将产品标识从“自由软件”改为“开放源码”，这个改变立刻赢得一片赞扬之声，以至于每个参与其中的人都觉得有些吃惊。

20世纪90年代中期，黑客文化中的实用派不断发展壮大，并逐渐演化为多个中心，一些相对独立的开发社区有了自我意识和魅力领袖，开

始在UNIX/Internet树干上蓬勃发展，继Linux之后，Larry Wall创立的Perl社区是其中最重要的一个文化，建立在John Osterhout的Tcl和Guido van Rossum的Python之上的文化，虽然稍小但也很有影响力。这三个社区都有自己设计的不同于GPL的版权方案，以表达其独立的意识形态。

3.3　理论宽松，实践严格

然而，在经历这些变化之后，人们对什么是“自由软件”或“开放源码”仍有着普遍认可的共识，在很多开源许可证中都能发现对此共识的清晰表达，其最关键要素都是一致的。

1997年，“Debian自由软件准则”提炼了这些共同要素，并形成了开放源码定义（OSD，参见http://www.opensource.org）。

定义指出，开源许可证必须保护任何个人或团体无条件修改开源软件（以及发布修改后软件版本）的权利。

所以，OSD（以及与OSD一致的版权声明，如GPL、BSD许可证、Perl的艺术许可证（Artistic License））隐含的规则是“任何人能干任何事”（anyone can hack anything），没有任何事情可以阻止人们获取任意开源产品（如自由软件基金会的gcc编译器）、复制其源码、推进其向不同方向演进，并都可声称是该产品。

这种演进上的分化称为“分支”(fork)，分支最重要的特点是它派生出一个随后不能交换代码的竞争项目，并导致开发社区潜在的分裂。（有的情况看上去像分支但其实不是，如Linux存在的多种发布版本。这种伪分支可能会导致不同的项目，但是它们使用的代码几乎相同，并且可以互相受益于对方开发的所有成果，它们在技术上和社会学上都不是浪费，也不会让人感觉到是“分支”。）

开源许可证没有对“分支”做任何限制，更不用说“伪分支”了。人们

可能会说这暗中鼓励了分支，但实际上，“伪分支”比较常见，分支却几乎没有发生过。重大项目极少产生分化，如果有，也总伴随着重新命名以及大量的公开解释，很明显，在诸如GNU Emacs/XEmacs分化、gcc/egcs分化，以及从BSD派生出的各种分化中，分化者都觉得他们在违背一个相当强大的社区准则[1]。

事实上，和“任何人能干任何事”共识相矛盾的是，开源文化有一套严格的但主要是“不允许”类型的所有权惯例。

这些惯例决定了谁能修改软件、在什么情况下可以修改，以及（特别是）谁有权利向社区发布修改后的版本。

文化中的一些禁忌凸显了这些准则，我们在此总结其中一些重要的内容，以便后面使用。

· 分化一个项目会遇到强大的社会压力，只有在极为必要的情况下才使用，而且要重新命名和做出大量的公开解释。

· 在没有项目主持人认可的情况下发布更新是令人不悦的，除非是特殊情况（如本质上不重要的移植bug修复）。

· 在项目历史、致谢表或维护列表中移除某个人的名字是绝对不可以的，除非当事人明确表示同意。

在本文的余下部分，我们将仔细研究这些禁忌和所有权惯例。我们将不仅探究这些概念是如何运转的，还将揭示开源社区背后隐藏的社会动力学及激励结构。

3.4 所有权和开放源码

在资产可以被无限复制和轻易改变，且其环境文化既不是强制权力关系也不是物质稀缺经济的情况下，“所有权”该如何理解呢？

实际上，开源文化对这个问题有着简单的回答：一个软件项目的“所有者”就是在社区中众所周知的对软件版本改动有唯一发布权的那个人。

（在讨论“所有权”这一节中，我使用单数形式，因为所有项目都有一个“所有者”，但有的项目是由一个小组共有的，我们将在后面讨论小组的内部动力学。）

根据标准的开源许可证，在软件演化的游戏中所有人一律平等。但实际情况是，“官方”补丁和“流氓”（rogue）补丁有着公认的差别：“官方”补丁由公众认可的维护者发布，人们会接受它并将它合并到软件中；而“流氓”补丁由第三方发布，它很少见，也常常不被信任。[2]

公开发布是很重要的事情，这很容易理解。开源社区通常鼓励人们因个人需要而给软件打补丁，而且也不关心在一个封闭的用户圈子或开发者圈子内的变更发布，只有当改动被经常性地发布到开源社区中，并和原版造成竞争时，“所有权”问题才会产生。

通常有三种方式获得开源项目的所有权：第一种也是最显然的，就是去创建这个项目，当这个项目在开始时就只有一个维护者而且这个维护者仍然起作用的时候，所有权问题是连提都不该提的。

第二种方式是获取前任对所有权的移交（有点像“接力棒传递”）。这在社区中很容易理解，当项目“所有者”不愿意或者不能在开发和维护中投入必要的时间时，他（她）有义务将项目移交给一个有能力的继任者。

如果是重大项目的控制权移交，做一个隆重的公开声明是很有意义的。虽然没有听说过开源社区能够实际干涉所有者对继任者的选择，但习惯做法上显然认为合法性公开是非常必要的。

对于较小的项目，如果发生所有权变更，通常在项目发布的变更历史中说明就可以了。如果控制权的交出在实际上并非出自前任的自愿，通常认为，在一个合理的时间段内，前任可以通过公开反对来获取社区支持并最终夺回控制权。

第三种方式是一个项目需要维护但项目所有者已经消失或失去兴趣了。如果你想维护该项目，你的责任是努力找到这个“所有者”，如果找不到，你可以在相关场所（比如Usenet上专注于该应用领域的新闻组）声明该项目似乎是一个“孤儿”，而你想为之负责。

习惯上你需要等一段时间后再声明你自己是新的所有者，在这个时期内，如果有人宣称他们事实上在为这个项目工作，那么他们的声明有效而你的无效。一般来说，向公众声明你的意图应该超过一次，比较好的做法是在多个相关论坛（以及新闻组、邮件列表）上发布声明并耐心等待回应，在等待前任所有者或其他权利人的响应上，你所做的看得见的努力越多，在没有回应时你的声明就越有效。

如果你通过了这个过程（在用户社区的见证下）并且没有任何异议，你就可以声称自己拥有这个孤儿项目的所有权，并将之写入history文件。然而，比起所有权交接，这种方式稳妥性略差，在你做出实质性改进（在用户社区的见证下）之前，你不能认为自己有完全的合法性。

我观察开源软件中这些习惯做法已经有20年了（可回溯至前FSF的古老年代），这些习惯有一些非常有意思的特点，其中最有意思的是，绝大多数黑客在并不了解习惯做法的情况下就知道去这样做。事实上，本文大概是第一次有意识和比较完整地总结了这些做法。

另一个有意思的特点是，他们对这种习惯做法的遵循，表现出非凡的（甚至是让人震惊的）一致性。我注意过足有数百个开源项目，不管是亲眼看到的还是听说的，严重违反传统的事情屈指可数。

第三个有意思的特点是，这些习惯会随时间演进，并有着连贯一致的趋势，这个趋势鼓励更多的公众责任心、更多的公众注意，以及对项目名誉和变更历史维护的更多关心，其目的是建立和巩固项目当前所有者的合法性。

这些特点表明习惯做法不是偶然产生的，而像是某种内在程序或生成模式（generative pattern）的产物，这些程序或模式对开源文化起着至关重要的作用。

早先曾有本文的评论者指出，比较互联网黑客文化和骇客/盗版文化（如“warez d00dz”，它主要围绕游戏破解和盗版专题BBS）有助于弄明白两者的生成模式。本文的后面部分将比较开源和d00dz。

3.5　Locke及土地所有权

若要理解开源这些习惯做法的生成模式，可在历史上找到一些类比，虽然这远远超出了黑客们通常关注的领域。如果你学过法律史和政治哲学，可能会辨识出开源的所有权理论在实质上等同于英裔美国人关于土地所有制的习惯法（common law）理论。

在这个理论中，有三种办法获得土地所有权：

如果在边远地区有无主的土地，人们可以通过开垦、劳作、筑起护栏并保卫这片土地，从而获取对这片土地的所有权。

对于已经有主的土地，所有权转移的常规方法是契约转让——也即从上任所有者那里获得契约。“契约链”在土地所有权理论中是很重要的：拥有一片土地的理想证据是一连串的契约移交，一直追溯到土地最早开垦的时候。

最后，习惯法理论注意到土地所有权可能被遗失或丢弃（如某片土地的所有者离世且没有继承人，或者某片土地无人认领且缺乏对应的契约文件）。这种情况下，某片土地成为遗弃土地，它有可能被“逆权侵占”（adverse possession）——有人搬进来、修缮它、保卫其所有权，就像是开垦了它那样。

在中央集权不存在或较弱的情况下，正如黑客惯例那样，习惯法理论也在有机地演变。在斯堪的纳维亚民族和日耳曼民族，它发展了一千多年。英国近代政治哲学家John Locke将其进一步系统化和理性化，所

以该理论有时也称Locken财产权理论。

逻辑上讲，如果在某个地方存在高度经济价值或生存价值的财产而没有一个强大到可以强制集中分配稀缺资源的唯一权力机构，类似的理论就会被发展出来。即便是在狩猎采集（hunter-gatherer）文化中，这一点也是正确的，虽然有时人们会浪漫地认为该文化没有财产的概念。举例来说，生活在非洲南部Kgalagadi（先前称Kalahari）沙漠的!Kung San布须曼人，虽然没有猎场所有权一说，但水洼或泉水是有所有权的，且遵循着类似Locke的理论。

!Kung san这个例子很有启发性，因为它表明Lockean财产权习俗仅仅产生于资源期望价值超出保卫成本的地方。猎场之所以不是财产，是因为从猎场获取的收益不可预期而且易变，猎场也不是日常维生的必需品（尽管被高度重视）；而水洼对生存至关重要，而且由于比较小而容易守卫。

本文标题中的“心智层”（noosphere）是思想上的领土，是指所有可能想法的空间[3]。我们在黑客传统中看到的只是Lockean财产权理论在心智层一个子集（也即所有程序的空间）上的体现。因此所谓“开垦心智层”，就是每个开源项目创始人所从事的活动。

FaréRideau（fare@tunes.org）正确地指出，黑客的工作领域并不完全是纯粹的思想领域，他认为黑客所拥有的“编程项目”，不仅包括内涵概念上的具体劳动（如开发、服务等），还包括与之相关的名声、信任等等。他继而指出，黑客项目所位于的空间不是心智层，而更像是一种

双重心智层空间，也即在心智层上探索的程序项目空间。（借用天文物理学概念，从词源学上讲，应该将这种双重空间称为“能层”（ergosphere）或“工作层”（sphere of work）。）

实际上，心智层和能层之间有什么区别，并不太影响我们前面的观点。FaréRideau所强调的纯粹意义上的心智层，是否真的存在还值得怀疑，估计只有柏拉图式的哲学家才会相信它。而只有在某人想说明人们不能拥有“想法”（心智层的元素），但可以拥有“想法的实例”（如项目）时，区分心智层和能层才有点实际意义。这将引发知识产权（intellectual property）理论的讨论，并超出了本文的范畴[4]。

为避免可能的困惑，这里需要声明一下：无论心智层还是能层，都和整个虚拟的电子媒介空间不同，后者有时称为“电脑空间”（cyberspace）(大多数黑客都厌恶这个称呼)，电脑空间中的资产按照完全不同的规则进行管理（在本质上更接近于物质层），一个人如果拥有了电脑空间所寄生的媒介或机器，就拥有了这部分电脑空间。

Lockean理论的逻辑容易让人想到，开源黑客之所以遵循这些传统，为的是保卫他们劳动付出的某种预期回报，回报必须大于在项目开垦上的投入、对版本历史（用来记载“契约链”）的维护成本、对孤儿项目“逆权侵占”前做出公开声明并等待的时间投入。

更进一步，开源的“收益”肯定不仅包含对该软件的使用，而且包含那些会被“分支”危害或稀释的东西。如果软件使用是唯一的收益，那么就不会存在对“分支”的禁忌，开源所有权也就不会那么类似于土地所有

权。而事实上，现存的开源版权中确实存在这种类型（也即“收益”仅仅为使用，“分支”并不是问题）。

某些“收益”是立刻可以被排除的：由于你不能通过网络连接来强行控制别人，所以不可能是为了追求权力；同样，开源文化中并没有很像金钱或本质上是稀缺经济的东西，所以黑客并不是在追求很接近物质财富的东西（比如收集稀缺的纪念品）。

开源活动确实存在一种帮助人们变得更有钱的可能，但也只是对这种可能提供有价值的线索而已。有时，某人在黑客文化中获得的声誉会在真实世界中产生经济意义：比如带来一个更好的工作机会、一份咨询合同或一纸出版协议。

这种额外作用对于大多数黑客是非常少见和非主流的，它远不能成为黑客动机的唯一解释——即便我们无视黑客们的反复声明：他们做这些不是为了钱，而是出于理想或爱。

然而，如何导致这种经济上的额外作用是值得研究的。下面我们会看到，如果能够理解开源文化自身的“声誉”动力，将能很好地解释这些。

3.6 黑客境遇和礼物文化

为理解“声誉”在开源文化中扮演的角色，让我们从史学移步至人类学和经济学，考察一下“交换文化”和“礼物文化”的区别。

人类对社会地位的竞争有一种天生的内驱力，它通过进化根植于人心。灵长类整个历史的90%都处于农业尚未被发明的年代，我们的祖先生活在小型的游牧型狩猎采集群体中，位居高位的个体（即那些能够最有效组成联盟并说服他人与自己合作的人）获得最健康的伴侣和最好的食物，对地位的驱动力会以不同的方式表达，并主要取决于生活必需品的稀缺程度。

大多数的人类组织模式都是为了适应稀缺和匮乏，每种模式有其不同的社会地位获取途径。

最简单的模式是“命令体系”。在命令体系中，稀缺物品被中心化的权力分配并以武力为后盾。这种体系的延展性很差[5]，当规模变大时，它变得越来越官僚和低效率。出于这个原因，如果命令体系的规模超过一个大型家族，那么它几乎总是寄生在另一个不同类型的更大的经济体之上。在命令体系中，社会地位的获取主要取决于对强制力量的使用。

我们的社会显然是一个“交换经济”，这是一种对稀缺性的微妙适应。与命令体系不同，它的延展性相当好，稀缺物品的分配主要是通过贸易和自愿合作（在事实上，竞争意愿的主要效果是产生合作行为）这种非中心化的方式。在交换经济下，社会地位主要取决于你有多少资源

（并非只是物质资源）可使用或交易。

大多数人同时存在这两种心理模式并理解它们之间如何交互。政府、军队和犯罪组织（举例来说）都是命令体系，它们都寄生在一个更大的交换经济体（我们称之为“自由市场”）上。然而，还有一种大多数人（除人类学家外）不太了解的和上述两类完全不同的模式：礼物文化。

礼物文化并不是对物质稀缺的适应，而是对物质充裕的适应。产生于没有生活必需品稀缺问题的人群中，在气候宜人且物产丰富的生态环境中，我们经常可以在其原居民文化中观察到礼物文化。我们也能在自身所处的社会中观察到这点，特别是娱乐行业和富豪阶层。

充裕性会使命令关系难以维持，会使交换关系变成无意义的游戏。在礼物文化中，社会地位并不取决于你控制了什么，而是你给予了什么。

所以才会有夸扣特尔族（Kwakiutl）酋长们的散财宴[2]，才会有千万富翁们精心准备的并常常是公开展示的慈善行为，才会有黑客们编写高质量开源代码的不懈努力。

用这种方式考察开源黑客社会，可以很清楚地看出它就是一种礼物文化。这里没有非常稀缺的“生存必需品”（如磁盘空间、网络带宽、计算能力），软件则是自由共享的，这种物质充裕导致成功的唯一标准就是同侪中的声誉。

然而，这些并不足以完全解释黑客文化的种种特性。在相同的（电

子）媒介上，骇客和warez d00dz也有着繁荣的礼物文化，但他们有完全不同的行为。他们文化中的群体意识比黑客的更强烈也更排他，他们更喜欢私藏秘密而不是分享秘密，容易看到，破解组织更多是分发没有源码的可执行文件，而不是给出破解技术。（对此行为的一种内部观点，见书后注释5）

这表明礼物文化有着不止一种的运转方式（虽然并不明显）。这关乎到历史和价值观，我在“黑客圈简史”一文中已经总结了黑客文化的历史，它产生馈赠行为的方式并不神秘。黑客通过对竞争形式的一系列选择，已经定义了他们的文化。本文的余下部分将会检视这些形式。

3.7 黑客的乐趣

在分析“声誉竞争”前顺便提一下，我并不是要贬低或忽视这种纯粹美学上的满足：设计优美的软件并让它运行。黑客们都经历过这种满足并乐在其中。如果某人没有这种意义上的动力，他根本就不可能成为一名黑客，正如不喜欢音乐的人永远不会成为作曲家一样。

所以我们也许该考虑下另一种黑客行为模型，在这种模型下，精湛工艺（craftsmanship）带来的单纯快乐是黑客首要的动机。“工艺”模型将黑客惯例解释为工艺机会和作品质量最大化的手段，这是否与“声誉竞争”模型的解释冲突？或者揭示出更多不同的结果？

这未必真实。为深入分析“工艺”模型，我们将重新审视是什么迫使黑客文化像礼物文化那样运转？在没有质量评判标准的情况下，人们如何将质量最大化？在稀缺经济并不适用的情况下，除了同侪评价还能怎样度量软件质量？看样子任何“工艺”文化最终必须将自己置身于“声誉竞争”的结构之中——并且，从事实上讲，观察历史上多个从中世纪行业协会发展出来的工艺文化，可以清楚地看到声誉竞争这种动力结构。

从另一个重要方面来看，“工艺”模型比“礼物文化”模型要弱一些，因为它并不有助于解释我们在本文一开始提到的矛盾。

最后，“工艺”动机本身可能不像我们所认为的那样能在心理上和“声誉竞争”划清界线，设想一下你优美的程序被锁进抽屉并且不再被使用，再设想一下它被很多人满意地使用着，哪个想法会给你带来满足？

不过，我们会继续关注“工艺”模型，很多黑客在直觉上接受它，而且它能够很好地解释一些个体行为[6]。

本文第一版在互联网上发布后，一位匿名读者评论道：“别为名声工作，如果你做得好，名声将伴随结果而来”。这是一个微妙而重要的观点，不论黑客是否注意到，声誉激励都将起到作用；因此，说到底，不管一名黑客是否理解他的行为属于声誉竞争，他的行为都会受此模型影响。

另一位读者则引用了Abraham Maslow关于人类动机著名的“价值体系”模型[7]，并认为“同侪尊重”激励和“编程乐趣”在层次上高于生存需要。从这个角度看，“编程乐趣”是对自我实现或自我超越需求的满足，这种高层次需求只有在低层次需求（包括生理安全、归属感、同侪尊重）被最低程度满足后，才会持续地表现出来。所以，“编程乐趣”要在一定社会环境下才能成为个体的主要动机，而“声誉竞争”可能对该环境提供起到了至关重要的作用。

3.8 声誉的多面性

追求同侪中声誉（声望）的意义何在？下面总结了适用于多种礼物文化的一般性原因：

首先，也是最明显的一点，在同侪中拥有好名声是一种最基本的激励，出于前面提到的进化上的原因，我们都渴望体验它。（一些人试着将这种对声誉的内驱力做各种升华，使其看起来和同侪没有那么明显的关联性，比如“荣誉”、“道德操守”、“虔诚”等等，但这并不能改变其内在本质。）

第二，声誉是很好的吸引他人注意和合作的途径（在纯礼物经济中，这是唯一的途径）。如果一个人慷慨、智慧、公平交易、有领导能力，或者有其他优秀品质，就更容易让人相信和他合作能获得好处。

第三，如果你所处的礼物经济和交换经济或命令体系互相关联交织，你的名声就可能传播到后两种环境中，使你在那里获得更高的地位。

除这些一般原因外，比起“真实世界”里的礼物经济，黑客文化中的一些特殊情况使声誉有着更高的价值。

主要的“特殊情况”是黑客所赠予的礼品（或换言之，那些明显是作者花费精力和时间做出的东西）非常复杂，与实物礼品或交换经济中的金钱相比，其价值体现很不明显，也很难客观区分其礼品的好坏，所以，黑客的地位竞争能否成功，很微妙地取决于同侪们挑剔的评价。

另一个特殊之处是开源文化相对单纯。绝大多数礼物文化都会有折中——不论是对交换经济关系的折中（如奢侈品交易），还是对命令经济关系的折中（如家庭或宗族结盟）。开源文化中不存在真正意义上类似的折中，也就是说，要想获取地位，除了同侪声誉，没有什么更多途径了。

3.9 所有者权利和声誉激励

现在，我们可以把前面关于黑客所有权习惯的分析整合成一个连贯的解释了，我们已经了解心智层开垦的产出，就是黑客礼物文化中的同侪声望以及所有它带来的二次收益和额外作用。

从这个理解出发，我们可以把黑客所沿袭的Lockean财产权习惯看做是一种将声誉激励最大化的手段——确保同侪将名誉赋给应得之人，而不会赋给不该得到的人。

这样，就很容易明白我们前面提到的那三个禁忌了。如果一个人的作品被他人占用或者乱改，他的声誉就会受到不公平的损害；禁忌（以及相关习惯）倾向于阻止这种情况发生。（或者更实际地看，黑客通常阻止项目产生分支或者出现流氓补丁，之所以否定其合法性，是为了防范同样的事情发生在自己身上。）

· 项目产生分支是不好的，因为分化前的项目贡献者会面临声誉风险，若要控制该风险，他们只能在分化后的两个子项目上同时保持活跃。（通常这是不现实的，因为它让人困惑或难以实施）。

· 发布“流氓”补丁（或者更糟糕的“流氓”二进制文件）会让项目所有者陷入声誉风险，即便官方代码是完美的，所有者仍然会因补丁中的bug而被抨击（见书后注释4）。

· 偷偷将某人的名字从项目中移除，是黑客文化中最极端的恶行。这相当窃贼偷盗了受害者赠予的礼物，并说是他自己的。

当然，分化一个项目或者发布“流氓”补丁也是对原先开发团队的声誉的直接攻击。如果我对你的项目这样做，就好像我在说“你做错了，你不能承担这个项目，而我可以”，并且任何使用我给出分支的人，都相当于支持这个挑战。但是，这本身是符合游戏规则的，虽然极端了点。它是最尖锐形式的同侪评价，它本身尚不足以构成禁忌，但毫无疑问是一种攻击。

这三种禁忌行为不但伤害了当事人（团队），也伤害了整个开源社区，因为它使得每个潜在的贡献者都觉察到馈赠行为或生产行为获得奖励的可能性变低了。

这里有必要补充一下其中两个禁忌的另一种解释。

黑客厌恶项目分化的另一个原因是，他们惋惜那些被浪费的重复工作——分化后的两个子项目总是有着或多或少平行的演化路线。他们也会注意到分支倾向于分裂合作开发者社区，使得两个子项目的人手都比父项目的人手更少。

一位读者曾指出，分支很少能有一个以上的后代存活下来（活下来是指能长期拥有一定的“市场份额”），这促使项目所有参与方合作并避免分化，因为如果产生分化，很难预先知道谁会落败，一旦落败，就只能看着他们曾经大量的工作完全消失或者默默凋零。

他还指出一个不争的事实，即分支很容易产生争论和对抗，这足以引发对项目团队的社会压力。争论和对抗都会妨碍团队合作，而团队合作正是每个贡献者要达到自己目标所必须的。

讨厌“流氓”补丁的另一个常见解释是：“流氓”补丁会产生子版本间的兼容问题，会使bug跟踪变得异常复杂，会额外增加维护者的工作量，因为维护者捕捉自己错误的工作量就已经够多了。

这些解释有一定道理，并且在一定程度上增强了Lockean财产权逻辑。在具备理性吸引力的同时，它们未能解释在偶尔发生禁忌违背或破坏时，为什么会有这么多的情感冲突和所有权防御——不仅受害者如此，那些旁观者和评论者也常常反应激烈。“重复工作”和“维护烦恼”这种相对理性的分析，很难解释这些现象。

至于第三个禁忌，除了“声誉竞争”模型，很难看到其他有解释力的理论。多数分析也只停留在“这不公平”的层面，我们将在下一节揭示它的内涵。

3.10 “自我”的问题

在本文开头部分，我曾提到文化中具有自适应能力的无意识部分往往与其外显的意识形态有所不同。其中最主要的一个例子体现在Locken所有权惯例上，尽管这些惯例违背了标准许可证所表述的意图，但仍然被广泛地遵循。

在与黑客讨论“声誉竞争”模型时，我观察到另一个有趣的现象，那就是很多黑客抵制这个模型，他们非常不愿承认驱动自己行为的动力来自于“对同侪声望的渴望”或是“自我的满足”（这是我当时未考虑后果而贴上的标签）。

这揭示了黑客文化一个有趣的方面，它有意识地不信任或者看不起“自我主义”或者基于自我的动机。“自我推销”往往会遭到批判，即便整个社区可能从中获得好处。正因如此，黑客文化中的“大人物”和“部落长者”需要言谈轻柔和幽默自贬来维护他们的地位。这种否定“自我”的态度和几乎完全建立在“自我”之上的激励结构是如何协调的？这太需要有一个解释了。

一个重要原因是欧裔美国人对“自我”通常所持的否定态度。大多数黑客所处的社会环境都教导他们对自我满足的渴求是不好的（或至少是不成熟的）动机；“自我”在最好情况下也只是用来形容“首席女高音”（prima donna）[3]的怪脾气，糟糕的情况下则会被认作精神病的征兆。只有把“自我”升华或伪装成“同侪声誉”、“自尊”、“专业素养”或“成

就感”时，人们才愿意接受它。

我完全可以另写一篇文章分析这些根植在我们文化遗产中的不健康观念，以及认为我们真的会有“无私”动机（尽管违背所有心理上和行为上的证据）这种自我欺骗式信念给我们带来了多么令人震惊的危害。要不是弗里德里希·威廉·尼采（Friedrich Wilhelm Nietzsche）和艾茵·兰德（Ayn Rand）已经完全将“利他”解构为未被意识的某种“利己”（且不论两人在其他方面的不足），也许我会乐意做这件事。

我并不是在这里谈伦理学或心理学，我只是观察到“自我有害”这种信念带来的一种较轻的危害，那就是它使一些黑客在有意理解他们自己文化的社会动力学时产生了情绪上的障碍。

让我们继续回到这条线索上的考察。由于黑客（子）文化对“自我”驱动行为有着强烈的抵制，人们不得不怀疑黑客是不是有一种特殊的自适应功能。要知道，这些禁忌在其他礼物文化中要微弱得多（或者不存在），比如在戏剧圈或富人圈的同侪文化中。

3.11 谦逊的价值

在确立“威望”是黑客文化奖励机制的核心后，我们需要理解为什么它看上去如此重要，以至于人们仍然遮遮掩掩并相当程度上不认可这个事实。

和盗版文化进行比较是有启发意义的。在盗版文化中，追求地位的行为是公开的甚至是喧闹的，这些破解者追求发布“zero-day warez”（在正版软件上市当天就发布出来的破解版本）时人们的欢呼，但他们对如何做到这点却闭口不谈。这些魔术师不愿泄露他们的秘密，整个破解文化的知识库只能是缓慢地增加。

相比之下，在黑客社区中，一个人的作品就是他的宣言。这里有着严格的精英意识（技术最好的人胜出），这里的信条是让质量说话，让黑客最自豪的是代码“好使”（just works），是让任何称职程序员都能看到的好东西，所以，黑客文化的知识库增长迅猛。

因此，“自我表现”禁忌会促进生产力，但这只是一个次生(second-order)效果；该禁忌直接保护的是社区同侪评价系统的信息质量，之所以要抑制自我吹捧或妄自尊大，是因为它像噪声一样，往往会破坏在创造性和合作性行为实验中得出的重要信号。

出于非常类似的原因，抨击作者而非代码是不合常规的，这一点微妙而有趣，黑客们会没有顾忌地在意识形态或个人差异上互相攻击，但从未听说有哪个黑客曾公开攻击另一个人的技术能力（即使是私下的非

议也很少见，如果有，也往往以柔和的方式表达），排错和差评总是针对项目而不是个人。

更进一步，人们并不会下意识将过往的bug归咎于开发者，普遍认为“bug已被修复”比“这里有过bug”更重要。就像一位读者所说，一个人是通过修复Emacs的bug获得地位，而不是通过修复Richard Stallman的bug获得地位——用已经修复的Emacs的bug来批评Stallman会被认为是极其恶劣的行为。

比较有趣的一点是和学术圈相比。在学术圈，公开批评他人有缺陷的作品是赢得声誉的重要方式，而类似行为在黑客文化中会被严厉禁止。由于如此严厉以至于我无法找到此类行为的任何材料，直到本文首次发表接近整整一年后，才有一位读者以不寻常的角度予以指出。

攻击他人能力的禁忌（学术圈没有这个禁忌）比起自我表现禁忌（这一点学术圈也有）更有揭示意义，因为我们可以将其关联到学术圈与黑客圈在沟通和支撑结构的差异上。

黑客文化中赠送礼物的媒介是无法触摸的，他们表达情感细节的沟通渠道十分贫乏，面对面的接触是非常规的，这使得比起其他礼物文化，黑客文化对噪声更缺乏忍耐力，对自我表现禁忌和能力攻击禁忌的解释也需要花更多功夫。任何稍具影响力的针对黑客能力的攻击事件，都会对声誉计分板造成不可容忍的干扰。

噪声所带来的危害也解释了为什么黑客社区的部落长老们需要在公共场合表现得谦逊。他们必须要看起来不自夸也不故作姿态，以便维持

这种抵制危险噪声的禁忌。[8]

谈吐柔和也是有用的，如果某人希望成为一个成功项目的维护者，他必须让社区信服他良好的判断力，因为维护者的主要工作是判断他人的代码，谁愿意将代码贡献给一个明显不能正确判断他们自己代码质量的人？或者一个试图从项目中沽名钓誉的人？潜在的贡献者希望项目领导人在客观采用他人代码时，能够谦逊而有风度地说："是的，这个的确比我的代码好，就用这个了"——然后将荣誉给予应得之人。

谦逊在开源世界里还有其他好处。很少有人想给人以项目"已经结束了"的感觉，因为这会使潜在贡献者觉得项目不需要自己。对项目保持谦虚可充分利用杠杆作用，假如有人展示了自己的优秀代码，然后说："唉，它没有x、y和z，所以还不是那么好"，通常，针对x、y和z的补丁很快就会出现。

最后，我个人曾观察到一些顶级黑客的自谦行为，表明了他们对个人崇拜的真实恐惧（这并非没有道理）。Linus Torvalds和Larry Wall都明白无误地展现过很多次对个人崇拜的回避。在一次集体外出晚餐的路上，我对Larry Wall开玩笑说："你是这里的第一黑客，你来挑餐厅。"他很明显地表现出了退缩。这是正确的，很多志愿者社区就是因为不能辨识共享价值观与领导者个性之间的差异而毁掉的，Larry和Linus肯定都已充分意识到这点。而另一方面，大多数黑客都很愿意有Larry遇上的这种问题，如果他们自己承认的话。

3.12　声誉竞争模型的推论

声誉竞争模型还可以给出更多的启示，有些可能不是一眼就能看出来，但大多数可以从事实中归纳出来：项目创建会比项目合作让人获得更多声誉；相比增量改进现有软件的“me,too”（译者注：仿造）产品，创意十足的产品会让人获得更多声誉；一个软件如果只有作者本人才理解或需要，则在声誉竞争中几乎连参赛资格都没有；对现有项目做出贡献会比另起一个新项目更容易吸引注意力；和成熟项目进行竞争比填补领域空白要难得多。

因此，存在一个和邻居们(最相似的竞争项目)的最佳距离。如果太近，会被认为是一个价值有限的“me,too”项目，是一个低劣的礼物（这样还不如为一个现有项目做贡献）；如果太远，就没有人会去用、去理解、去感知作者的努力（仍然被认为是低劣礼物）。这造就了一种开垦心智层的模式，即一种更像是移民们分散开来向地理边缘拓展的模式，而不是随机的扩散受限的分形图式。项目总是倾向于填充前沿地带的功能性缺口（书后注释9进一步讨论了新颖性的诱惑力）。

一些非常成功的项目成了类别杀手，没人希望在它周围安营扎寨，因为和这些项目的基础相比，新项目很难再吸引黑客们的注意力。那些发现另起炉灶希望不大的人，转而会为那些大型成功项目增加扩展。类别杀手最经典的例子当数GNU Emacs，该项目及其变种在全功能可编程编辑器的生态环境中占据得如此彻底，以至于自20世纪80年代以来，甚

至都没有竞争产品能超越Emacs当年还只是单人作品时的水平。人们转而编写各种Emacs模式。

从全球看来，这两个倾向（“填补空白”和“类别杀手”）是开源项目发展的总体趋势。上世纪70年代的开源项目大多都是玩具和原型，80年代多为开发工具和互联网工具，90年代则转向操作系统。每当一个问题被基本解决后，黑客们就开始攻克更新和更难的问题。

这种趋势对未来有着很有意思的启示意义。1998年初，Linux看上去已经非常像是“开源操作系统”类别的杀手，那些原本可能写出与之竞争的操作系统的人，转而去写Linux设备驱动和扩展，多数较底层的（曾经被黑客们渴望的）开源工具已经有了，那还剩下什么呢。

是应用。在第三个千年的开始，我们大可预言开源会转向最后一块处女地——写给非技术人员的程序。GIMP（http://www.gimp.org ）是一个早期标志性项目，这个类似Photoshop的图像处理应用，是开源过去十年来第一款满足商业软件GUI界面标准的面向最终用户的重要应用。另一个则是引起很多关注的应用工具项目如KDE（http://www.kde.org）和GNOME（http://www.gnome.org）。

一位读者指出，将开源开发和土地开垦类比，也解释了为什么黑客对微软“拥抱并拓展”（embrace and extend）策略——将互联网协议复杂化，然后将其封闭——有着发自内心的愤怒。黑客文化可以和多数封闭软件共存，比如Adobe Photoshop的存在并不能明显减少黑客对GIMP（类同于Photoshop的开源产品）附近领域的兴趣。但如果微软能

够成功地将协议“去市场化”（de-commoditizing），那就只有微软的程序员才能写出符合该协议的软件了。微软不仅通过不断垄断伤害消费者，他们还在数量和质量上减少了可供黑客开垦和耕作的心智层空间[9]。怪不得黑客常把微软的策略称为“协议污染”，这种反应就好比农民看见有人往他们用来灌溉庄稼的河水里投毒。

最后，声誉竞争模型解释了一个常被引用的格言，即“自称是黑客不代表你就是黑客，只有其他黑客认为你是黑客，你才是黑客”[10]。从这个角度看，所谓黑客，是一个通过贡献礼物表现出他（或她）既拥有技术能力又懂得声誉竞争如何运转的人。对是否为黑客的判断是一种感知和认同，只能由那在文化中已经做得很好的人给出。

3.13　什么才是好礼物

如何评判礼物价值并给予相应的尊重，在黑客文化中有着始终如一的标准。主要如下：

1.如果它不能像我所预期的那样工作，那就不是好的——不管它多么聪明和有原创性。

注意“预期”这个词，这条规则并不是要求十全十美（beta测试和试验性软件是允许有bug的），而是要求：根据项目所处阶段和开发者对该阶段的说明，用户能准确地估计风险。

这条规则使得开源软件倾向于长期停留在beta版，开发者只有在确信软件不会有很多问题时，才会发布1.0版。开源世界的1.0版意味“开发者愿意拿自己的名誉赌它好使”，而闭源世界的1.0版则意味着“如果你很谨慎，不要用这版”。

2.在心智层的拓展性工作要比在某功能域内（对现有作品）的重复性工作好。

对这条规则的简单理解是：原创性工作比功能复制要好。但实际上不完全是，如果你复制的是闭源软件的功能，并借此破解了封闭的协议或格式，使得这个领域在开源世界里也可使用，那你的工作就和原创作品一样值得赞许。

比如，开源世界里最负盛名的项目之一是Samba——它使得UNIX机器可以作为客户端或者服务器运行微软私有的SMB文件共享协议。它并

没有多少创新性工作，而几乎都是在做一些繁杂的逆向工程。然而，Samba团队的成员被视作英雄，因为他们破坏了微软锁定整个用户群体的目的，并使其设立在心智层大片区域上的警戒线无效。

3.能进入主要发行版的作品比不能进入的好。在所有主要发行版中都包含的作品最令人尊敬。

主要发行版不仅包括大的Linux发行版如Red Hat、Debian、Caldera和SuSE，也包括其他一些发行版——如BSD发行版或自由软件基金源码集，由于发行者会维护自己的声誉，其质量也自然能得到保证。

4.“使用”是最真实的赞美，类别杀手比同类竞争者好。

信任别人的判断是同侪评价的基础。这很正常，因为没人有时间去仔细评判所有可能的选择，所以人们会认为：多数人使用的作品优于那些少数人使用的。

如果作品好到没人再想使用其他备选，作者将会获得巨大的威望。那些被最广泛使用的原创型类别杀手，会被纳入所有的主要发行版中，并获得最大可能的同侪尊重。成功做到这点超过一次的人，将会被人们半开玩笑、半认真地称为“大神”（demigods）。

5.相比那些只挑有趣和简单工作的人，长期致力于艰苦和乏味工作（如调试、写文档）的人更令人钦佩。

这条规则描述社区如何回报那些黑客天生不愿意做但实属必要的工作，某种程度上它和下面这条矛盾：

6.重要的功能扩展比低层次的修补好。

这条规则似乎是针对一次性工作的评价。相对于修补bug而言，给软件增加功能特性有可能得到更多回报——除非这个bug异常令人厌恶或者难以寻找，因为将这种bug找出来本身就证明了非凡的技术和才能。但当这些工作是长期行为的话，一个长期关注和排除bug（甚至是普通bug）的人，其地位要高于那些花费相近工作量在增加简单功能上的人。

一位读者指出，这些规则会很有意思地相互影响，并且不一定总能获得最大可能效用。如果问一个黑客，他是愿意因为一个全新工具还是因为扩展别人软件而为人所知，答案毫无疑问都是“新工具”，但如果问这两个选项：(a)一个隐藏在OS之下的全新工具，每天可能只会被用到几次，但一经问世便迅速成为一个类别杀手；(b)对一个现有工具的若干扩展，这些扩展既不是特别新颖也不是类别杀手，但拥有数量庞大的用户，扩展对用户可见且被用户每天使用。黑客在选择(a)之前可能会犹豫一会，因为这两种选择几乎是等量齐观的。

这位读者问我时还加上了一句：“(a)是fetchmail，(b)是你关于Emacs的若干个扩展，如vc.el和gud.el”，他说得没错，我更喜欢被称为“fetchmail的作者”而不是“很多Emacs模式的作者”，尽管历经时间变迁，后者可能会有更多的使用总量。

很简单，拥有全新“品牌标识”的工作比为已有品牌增添功能更能吸引注意力。这些规则以及它们所揭示的黑客文化计分板系统的内涵，会引出供进一步研究的很好话题。

3.14　心智层所有权和动物行为学

为理解Lockean所有权惯例的起因和后果，我们可以从动物行为学（特别是领土方面）的角度来看。

所有权是动物领土权的抽象，它之所以被进化出来，是为了减少物种内的暴力争斗。狼通过标记边界和尊重其他同类的边界，可以减少陷入争斗的机会，而争斗可能会使它虚弱或死亡，从而降低它成功繁殖的概率。类似的，人类社会的所有权是为了防止人类之间的冲突，通过设定边界，可以清楚地区分和平行为与侵犯行为。

在某些圈子里很流行把人类社会的所有权描述为任意的社会约定，这绝对是错误的。养狗人几乎都体验过动物领土权和人类财产权在本质上的连贯性——狗在陌生人接近主人财产时会吠叫。作为我们驯养的狼类亲戚，狗在本能上知道，财产权不仅仅是社会约定或游戏，而是至关重要的防范暴力冲突的进化机制。（从这点上讲，它比很多人类政治理论家聪明。）

所有权声明（就像领土标记）是一种述行式（performative）行为，一种宣布防御边界的方法。开源社区支持的所有权声明是一种摩擦最小化和合作最大化的方法，尽管“所有权声明”比栅栏或者狗吠要抽象得多，尽管有时只是在README文件中声明一下项目维护者的名字，但这个道理是没错的。它是领土权的抽象，并（像其他形式的所有权一样）基于领土本能，一种为协助解决冲突而进化出来的本能。

这种动物行为学上的分析，乍看上去很抽象，也很难和实际的黑客行为联系起来。但它能推导出一些重要结论，比如为什么Web站点会流行，特别是为什么一个开源项目有了自己的网站，就会比那些没有的看起来更“真实”和更实在。

客观分析的话，这似乎很难解释。因为与创建及维护项目相比（即便是很小的项目），做网页很容易，很难理解为什么有个网页就让项目看起来更重要或者更不一般。

Web自身的功能特点也不足以解释这个问题。Web提供的通信功能完全可以由FTP站点、邮件列表和Usenet组合起来实现。事实上，对项目来说，更常规的做法是通过邮件列表和新闻组进行沟通，而不是通过Web。那么，将项目安家在网站上的做法为什么会流行起来？

“主页”（home page）这个术语中暗含的比喻可以给我们一些线索。创立一个开源项目相当于在心智层上的发布一个领土声明（习惯上也认可这点），但并不是强迫人们在心理层面上接受它。因为不管怎么说，软件没有自然上的位置，并且可以很快地被复制。只有在经过一番努力之后，软件是“领土”和“财产”的观念才会融入我们的直觉认识中。

通过建立项目“主页”，使项目在组织上更有空间感的万维网王国中建立起“家园”（home territory），从而具象化了在程序空间中“开垦家园”的抽象概念。从“心智层”落到“电脑空间”并不能让我们得到真实世界中栅栏和狗吠的感觉，但它的确把抽象的财产声明和人们对领土的直觉认识更牢固地连接了起来，这也正是为什么拥有网页的项目看起来

更“真实”一些。

超链接和好的搜索引擎更强化了这点。一个有网页的项目，会更容易被探寻心智层邻居的人注意到。人们能够链接到它或搜索到它，所以，网页是一个更好的广告，一个更有效的述行活动，一个更有力的领土声明。

动物行为学分析也促使我们更近距离观察开源文化中的冲突处理机制。它使我们有理由认为，所有权惯例除了最大化声誉激励外，还有防止和解决冲突的作用。

3.15　冲突的起因

我们可以把开源软件中的冲突辨识为以下主要四类：

· 谁来做有约束力的决定?

· 谁该得到荣誉或责备，因为什么?

· 如何防范劳动成果被复制? 如何防范流氓版本使bug跟踪变得更复杂?

· 从技术上讲，什么是正确的事?

至于“什么是正确的事”，其实不太算个问题。因为对于任何这种问题，要么有一个所有相关方都接受的客观的决策方法，要么没有。如果有的话，问题解决，皆大欢喜。如果没有，它就会归入到“谁来做决定”这个问题。

与此对应，任何一种项目冲突解决理论，必须要解决三个问题：(a)谁来负责做设计决策？(b)如何决定哪个贡献者应该被授予荣誉，如何授予？(c)如何保持项目团队和产品不被分裂为多个分支？

所有权习惯在解决(a)和(c)上的作用是很清楚的，习惯坚持认为应该由项目所有者做最终决策。我们也已观察到，习惯反对因分支而引起的所有权稀释，并会对这类行为施加很大压力。

比较有意思的是，即便有人不考虑声誉竞争模型，而仅仅从纯粹的“工艺”模型分析，这些习惯仍然讲得通。他们认为，习惯更多是保护工匠（craftsman）按照自己眼光做出选择的权益，而不是为了防止声誉

激励变弱。

但是，工艺模型不足以解释和(b)相关的黑客习俗，即什么人因什么而得到荣誉——因为一个纯粹的工匠，如果对声誉竞争不感兴趣，将不会在乎这些。为了分析这些，我们需要将Lockean理论更推进一步，去考察项目内以及多个项目间的冲突及财产权运转。

3.16 项目组织结构和所有权

最一般的情况是项目只有一个所有者/维护者。这种情况下不可能有冲突，所有者做一切决定并独担所有的荣誉和责备。唯一可能的冲突是继任者问题——当所有者消失了或者不再有兴趣时，谁将成为新的所有者。社区此时还关心问题（c）即分支问题，这种关心在黑客文化中表现为移交准则：所有者/维护者不再维护项目时，应公开地将权利移交给某人。

最简单的非一般情况则是项目为“善意独裁者”[4]拥有，其下有多位共同维护者。习惯上讲，项目团队偏好使用这种方式，Linux内核及Emacs项目就很好地使用了这种方式，在解决“谁来做决策”问题上，该方式不比任何其他方式差。

通常，善意独裁者这种组织形式是由所有者/维护者组织形式（在创立者不断吸引贡献者的过程中）演变而来的。即便所有者仍然保持专断地位，这种形式也可能引入一定程度的争议：谁因为项目的哪个部分获得荣誉。

在这种情况下，习惯要求所有者/独裁者有义务公平地将荣誉给予贡献者（比如在README或history文件中适当地提及）。对Lockean财产模型而言，这意味着如果你对项目做出了贡献，你就会获得一份声誉上的回报（正面或负面的）。

按这个逻辑推理，我们明白善意独裁者事实上并不绝对拥有整个项

目。虽然他有权做出强制性决策，但实际上是通过交易一部分声誉回报来换取他人的工作。这让人不禁联想到与之非常类似的佃农耕作，当然这点除外：即便贡献者不再为项目工作，其名字仍然保留在名誉表上，并能继续“赚取”一定程度的声誉。

当善意独裁者项目的参与者不断增加时，它倾向于发展出两层级的贡献者结构：普通贡献者和合作开发者。成为合作开发者的一个典型途径是承担起项目主要子系统的责任，另一个途径则是成为负责识别和修复bug的“最高修复官”（lord high fixer）。这两种情况下，合作开发者都是做出了大量和持续时间投入的项目贡献者。

在我们的分析中，子系统所有者是一个特别重要并值得深入研究的角色。黑客们常说“责任背后是权力”，一个合作开发者在承担起维护某个子系统的责任后，通常有机会掌控子系统及其对外接口的实现，其决策仅受项目领导人（同时也是架构师）的修正。我们观察到该规则有效地在Lockean模型上圈起了项目财产，它正如其他财产边界线一样起到了避免冲突的作用。

从习惯上讲，如果项目有合作开发者，“独裁者”或项目领导人应该在关键决策上征询合作开发者的意见，尤其当决策关乎某个合作开发者所“拥有”（也即投入了时间并为之负责）的子系统时。一个有智慧的领导者，会认识到项目内部财产边界的作用，不会轻易干扰或推翻子系统所有者做出的决定。

一些非常大型的项目则完全抛弃了善意独裁者模型，其做法是将合

作开发者转为投票委员（如Apache），或是在资深合作开发者内部轮流掌权，Perl开发者就是这么组织起来的。

这种复杂的组织形式被广泛认为是不稳定和运行困难的。很明显这种困难主要来自于“委员会设计” [5]以及委员会本身可能产生的问题，这些都是黑客文化在意识上所理解的问题，但我认为，黑客之所以发自肺腑地对这种委员会或者轮流坐庄的组织方式感到不舒服，部分原因是这种方式很难和黑客们潜意识中的Lockean模型相适应，Lockean模型可以用于较为简单的情况，但如果用于这种复杂的组织形式，在分析所有权时，不论是控制意义上的所有权，还是声誉回报意义上的所有权，都有很多问题。在这种形式中，我们很难看到内部边界，并因此很难避免冲突，除非委员会内部享有极高水平的和谐与信任。

3.17　冲突和冲突解决

我们已经看到设计权的分布和所有权的分散会导致项目角色复杂性增加，虽然它是一种有效的激励分配机制，但同时也稀释了项目领导者的权力，稀释了领导者平息潜在冲突的权力。

有关设计的技术争议看上去很容易引起内部两败俱伤的争执，但它们很少真的能引起严重冲突，这些争议通常可较容易地由领土规则（也即“责任背后是权力”）解决。

另一个解决冲突的方式是“资深者胜”——如果两个或更多个贡献者有分歧，该分歧在客观上很难解决，并且谁也不拥有该分歧的领土权，那么在整个项目中投入工作最多的一方（也即在整个项目中拥有最多领土权的一方）胜出。

（对应地，投入最少的一方输掉。很有意思的是，很多关系型数据库也采用了同样的启发性方法解决死锁问题，当两个线程在资源上造成死锁时，在当前事务中投入最少的那个线程将会成为死锁受害者并被终止掉，而运行事务时间最长的或级别更高的，则会成为胜利者。）

这些规则通常足以解决大多数的项目争议，如果还不行的话，则由项目领导人来决断。以上这些办法都不起作用的情形非常罕见。

冲突通常不会变得很严重，除非两种评判标准（“责任背后是权力”和“资深者胜”）指向不同方向且项目领导人权力较弱或者缺位时。发生这种情况最显而易见的例子是项目领导人消失后的继承权争议，我

曾经历过一次这样的争斗，其过程令人厌恶、充满痛苦且旷日持久，直到所有相关方都耗尽心力而不得不交给外部人士处理时，问题才得到解决，我真心希望永远也不再和这种事扯上任何一点儿关系。

最后，若想让这些冲突解决机制生效，需要整个黑客社区愿意执行它们，唯一可用的执行机制是“批判”和“放逐”——公开谴责那些破坏习俗的人，并拒绝和他们再次合作。

3.18　文化移入[6]及学术界关联

本文的早期版本提出了这些问题：社区是如何告知和指导其成员遵循习俗的？习俗在半意识层面上是不是不证自明的或自组织的（self-organizing）？它们是通过事例还是通过明确的指示来传授？

明确指示是很罕见的，但愿仅仅是因为目前几乎没有什么可用的关于黑客文化准则的明确描述。

多数准则都是通过事例来传承的。举个很简单的例子，所有软件在发布时都有一个叫README或者READ.ME的文件，用于人们在首次浏览该发布版时获得指导信息，这个传统早在上世纪80年代初就已很好建立起来，甚至有时还会被写下来，但任何人只要看过一些发布版，就能够推理出这点。

另一方面，一旦人们对声誉竞争有一些基本的（也许是无意识的）理解，就会发现很多黑客习俗都是自组织的。大多数黑客根本不需要被告知本文前面提到的三个禁忌，或至少在被问及这些禁忌时，他们会说这些都是无须告知的，是不言自明的。这个现象吸引我们做进一步的分析——也许我们可以从黑客获取其文化知识的过程中找到解释。

一些文化使用隐藏的线索(更精确地说是一些具有宗教或神秘意味的“奥秘”)作为文化移入机制。它们是一些不对外公开的秘密，但要求那些充满渴望的新手们能发现和推理出来。一个人如果想被该文化接受，必须要展示他能理解这些秘密，而且以一种被文化认可的方式掌握

了这些秘密。

黑客文化非常有意识地大量使用这样的线索或测试。我们可以看到这个过程至少在三个层次上起作用：

· 类似密码一样的特定秘密。举例来说，有一个叫做alt.sysadmin.recovery的Usenet新闻组，其中有一些非常明确的秘密，如果你不知道这些秘密就无法发文，知道秘密则被认为有发文资格。其成员有着严厉的禁忌以防范秘密泄漏。

· 某种技术秘密的入门要求。一个人能给出有价值礼物的前提是，他（她）已经吸收了大量的技术知识（比如至少要了解一门主要的计算机语言），正如隐藏秘密是在细节层面提要求，这是在宏观层面提要求，它起到了素质（比如抽象思考、毅力、心理弹性等能力）过滤器的作用。想在黑客文化中发挥作用，这些素质是必需的。

· 社会语境下的秘密。人们是通过投身于某个特定项目而进入黑客文化的，每个项目都是一个鲜活的社会语境，想要成为项目的贡献者，一个人需要在技术上和社会上都对它进行研究和理解（具体来说，通常是通过阅读项目Web页和/或邮件记录来做到这点）。新手正是通过项目团队，从老手经历的事件中获得阅历。

在这个获得秘密的过程中，想成为黑客的人学到了语境中的知识，（随后）那三个禁忌和其他习惯也就不言自明了。

可能还有人认为，黑客礼物文化结构本身就是其核心秘密。一个人如果不能展示他（她）对声誉竞争及其相应习惯、禁忌以及传统有发自

肺腑的理解，就不会被文化接受（具体来说：就是没人认为你是黑客）。但这其实很普遍，所有文化都会要求渴望加入者有这样的理解，而黑客文化从未表示出他们想把其内在逻辑和风俗习惯当做秘密来保守——至少从未有人因为我揭露了这些而发火！

有无以计数的读者曾指出：黑客文化的所有权习惯看上去和学术界（特别是科学研究团体）的做法有着密切的关联（也许就是直接源于后者）。在开垦盛产创意的领地时，研究团体有着类似的问题，并展示出非常类似的适应性解决方法：同侪评价及声誉。

鉴于很多黑客是在学术环境下成长起来的（人们常常是在大学里学习黑客技能），所以了解一下学术界和黑客文化在适应模式上的相似程度，会很有助于理解这些习俗是如何应用的。

学术界有很多明显与黑客“礼物文化”（正如我前面所描述的）相类似的做法，一旦一个研究者获得终身职位，他就不用再担心生计问题。（事实上，终身职位概念也许可以追溯到更早的礼物文化，那时“自然哲学家”主要是生活富足的绅士，他们有大把的时间可以投入到研究中。）没有生存问题之后，名望就成了追求目标，它促进人们在期刊和其他媒介上分享新的创意和研究成果。这是很有客观价值的，因为科学研究正如黑客文化，都非常有赖于“站在巨人的肩膀上”，人们不必要一次又一次地重新发现基本原理。

所以，黑客习俗可能仅仅是研究团体惯常做法的映射，且（大多数情况下）黑客个体正是在那儿获得了这些习俗。不过这可能说得有点过

了，因为掌握这些习俗并不难，一个聪明的高中生都能很快学到！

3.19 礼物胜过交换

还有一些更有意思的可能性。我之所以推测学术界和黑客文化有着共同的适应模式，并不是因为它们有着基因上的关联，而是因为在自然法则和人类固有本能的条件下，它们都已经演化到其从事领域的最佳社会组织形态。自由市场经济是全世界范围内通过合作获得经济效能的最佳方法，这一点看来已成为历史定论，同样，基于声誉竞争的礼物文化可能是通过合作产生（和检验）高质量创造性工作的全世界范围内的最佳方法。

这个理论背后有大量关于艺术与报酬之间相互作用的心理学研究做支持[11]，这些研究没有得到应有的关注，部分原因可能是一些普及读物表现出对它们过度解读的倾向：认为它们是对自由市场和知识产权的攻击。不过，研究结果的确表明，某些类型的稀缺经济报酬，在事实上减少了创意工作者（如程序员）的生产力。

布兰代斯大学心理学家Theresa Amabile，在1984年发表的一篇关于关动机和报酬的研究中谨慎地总结道：“与完全出于兴趣的工作相比，被委托的工作通常表现出较少的创造性。”Amabile还观察到：“活动越复杂，就越容易被外部的报酬损害。”该研究还指出，固定工资不会降低人们的积极性，但计件工资和奖金会，这一点很有意思。

因此，给那些做汉堡和挖地沟的人们发放绩效奖金可能是经济正确的，但更明智的做法是：在编程工作中把薪水和绩效奖励完全分开，并

让人们选择他们自己的项目（这两种趋势自然而然地在开源世界里得到了体现）。这些事实表明：只有当程序员非常积极以至于没有奖励他（她）也愿意工作时，才是唯一应该给予绩效奖励的时候。

这个领域的其他研究者把目光投向了黑客们非常在意的自主性和创意自由度问题，“如果一个人越是感受到自主性受限”，罗切斯特大学心理学副教授Richard Ryan说，“其创造力就会越少。”

对任何一个任务，如果它更像是手段而不是它本身的话，往往会降低人们的积极性。即便是赢取比赛或获得同侪尊敬，如果觉得获取胜利只是为了求得回报，也一样会觉得没意思（这也许可以解释为什么黑客文化禁止那些毫不隐瞒的对尊重的追求和索取）。

这也使管理变得更复杂，支配性的语言反馈似乎也如计件报酬一样降低着人们的积极性。

Ryan发现，在公司里听到“很好，你就该这么做”的职员“比那些接收到具体反馈信息的人们明显缺乏内在动机一些”。

给予激励是聪明的做法，但一定不能有附加条件，以避免把事情搞糟糕。Ryan观察到以下两种说法有着完全不同的效果：“我给你报酬是因为我认可你工作的价值”和“你得到报酬是因为你达到了我的标准”，第一种说法不会挫伤积极性，但第二种会。

以这些心理学观察为依据，我们可以得出论断：开源开发团体在生产力上会远远超过（特别是长远地看，创造性作为生产力倍乘器的作用会越来越重要）同等规模与技能的闭源程序员团体，后者受稀缺性报酬

激励但同时也导致动力不足。

这从另一个略微不同的角度，重申了“大教堂与集市”中的论断：最终，当自由市场经济开始创造出足够的财富盈余时，大量程序员可以生活在后稀缺的礼物文化中，而软件产品的工业\工厂模式注定走向衰亡。

事实上，获取最高软件生产力的药方看上去自相矛盾而又颇具禅意：如果你想获得最有效率的产品，你必须放弃促进程序员生产力。做好他们的后勤，让他们自己做主，并忘掉最后期限。在传统的管理者看来，这种几近疯狂的宽容注定失败——但它的的确确有效，使用这种方法，开源文化正势如破竹地痛击着它的对手。

3.20 结论：从习惯到习惯法

我们已经研究了那些规范开源软件所有权和控制权的习惯，看到了它们所隐含的与Lockean土地权理论相类同的财产权理论。我们将这些习惯与黑客世界的“礼物文化”（参与者付出时间、精力和创意，以图在竞争获取声望的文化）联系了起来，并考察了它们对黑客解决冲突所带来的影响。

接下来很自然的问题是：“这有什么意义？”黑客们在形成这些习惯并遵循这些习惯时（直到现在）都未进行有意识的分析，这种有意识的分析能给我们带来什么实用的东西，并不是一下子就能清楚的——除非，也许，我们可以不再仅仅是描述它，而是给出解决问题的具体建议并推理出能让习惯变得更好的方法。

我们已经发现黑客习俗和英裔美国人普通法传统下的土地所有制理论在逻辑上非常类同。从历史上讲[12]，欧洲部落文化发明了这些传统，并改善了他们的冲突解决体系——从表达不清的、半意识的习惯体系发展到掌握在部落智者记忆中明确的习惯法，并最终发展成为成书面制度。

也许，当我们人数不断增加、当所有新成员的文化移入变得困难时，黑客文化也该做一些类似的事——将一些优秀实践发展成为规范，解决开源项目中可能出现的各种争议，并发展出一种可以让社区中资深成员调解纠纷的仲裁传统。

本文的分析将以前隐性的东西显性化，并大致勾勒了这种规范的轮廓。规范不能强加于人，它必须被各个项目的创建者或所有者自愿接受；由于作用在黑客文化上的压力会随时间变化，规范也不能全然僵化；最后，为了让这些规范实施起来，它必须能反映黑客部落的普遍共识。

我已经开始着手书写这些规范，暂且称之为“Malvern协议”（Malvern是我居住的一个小城市）。如果本文的大体分析能被足够广泛地接受，我会将Malvern协议公开，作为纷争解决的范例程式。如果有兴趣批评和帮助制定这套规范，或仅仅反馈这是否是个好想法，都请通过email联系我：esr@thyrsus.com

3.21　进一步研究的问题

黑客文化（包括我自己）已经发现，大型项目如果没有一位“善意独裁者”领导，往往很脆弱。大多数这种项目都失败了，少数几个则取得了令人惊叹的成功并变得很有影响力（Perl、Apache和KDE），没人真正懂得这其中的原因。人们隐隐约约觉得，这种项目都是独特的，其成功或失败，都建立在其特定成员形成的群体动力学之上，但这是真的吗？是否存在能让其他团队效仿的可复制策略？

[1] 心智层（noosphere，又译智力圈）是指人类思想的领域范围，或理解为人类智力活动的总和，在概念上与“大气层”（atmosphere）、“生命层”（biosphere）相类比。——译者注

[2] 夸扣特尔人是美洲印第安人的一支，居住在加拿大不列颠哥伦比亚省海岸地区和温哥华岛北部地区。夸扣特尔人的散财宴（potlatch）是一种庆祝宴会，通常是婚庆或就职庆典。主人会根据每个客人的不同身份或地位分发礼物。当存在竞争对手的时候，宴会往往伴随着炫耀性的斗富，比如大肆分发或破坏贵重物品，以显示自己在财富上的优越性。——译者注

[3] 首席女高音（prima donna，或译“布玛多娜”）这个称呼常用在歌剧剧团中，通常是在剧团中位于首席地位的女高音（类似于中国戏曲剧团中的“当家花旦”）。人们通常认为首席女高音是自我本位的，经常无理取闹的和脾气暴躁的，她们往往将自己的意见强加于他人意见之

上。——译者注

[4] 善意独裁者（benevolent dictator）是传统“开明君主”的现代叫法，是指独裁领导者运用自己的政治权力为人们造福而不是仅仅为了他(她)自己或者一小部分人的利益。——译者注

[5] 委员会设计（design-by-committee）是一个有讽刺和贬损意味的术语，用于指一组人（尤其是缺乏优秀的领导时）共同设计产生的东西（通常是技术系统和标准）。这种方式设计出来的产物常常过于复杂、内部不一致、有逻辑缺陷、平庸陈腐、缺乏一致的愿景。经常有人拿这种民主式设计与独立设计、独裁者设计做比较。

[6] 原文是Acculturation，指人逐渐适应新文化的过程。——译者注

4.魔法锅

本文分析了开源现象中不断演化的经济基础，首先驳斥了一些关于软件开发资助和软件价格结构的常见神话，然后给出了软件合作稳定性的博弈分析。我给出了开源开发可持续资助的九个模型，两个是非盈利性的，七个是盈利性的。然后本文给出了一种定性理论，来说明什么时候关闭软件项目在经济上是合理的。我研究了一些新的资助开源开发的盈利性机制，其中包括对资助体系和任务市场的新发现。最后，本文试着给出了一些关于未来的预测性结论。

4.1　与魔法无异

在威尔士神话中，Ceridwen女神有一只大锅，当她念出只有她自己知道的咒语时，这只锅就能魔法似地变出有营养的食物。在现代科学中，Buckminster Fuller给出了“以少成多”[1]的概念，指出技术会变得越来越便宜和有效，早期设计中需要投入的物理资源被越来越多地替换成信息内容。Arthur C.Clarke则指出：“任何足够先进的技术都无异于魔法。”

对很多人来说，开源社区的成功看起来像魔法一样不可思议，免费的高质量软件如果能不断生产出来固然很好，但在充满竞争和资源短缺的真实世界里，这能持续下去吗。开源是怎么做到的？Ceridwen女神的大锅仅仅是一个魔术把戏吗？如果不是的话，在开源世界里如何实现“以少成多”？——女神念动的是什么咒语？

4.2　礼物文化之外

开源文化的成功，的确让很多自以为了解软件开发的人感到大惑不解。“大教堂与集市”一文描述了分布式合作开发模式是怎样颠覆了Brooks法则，使得单个项目的可靠性和质量都能达到空前的水平。“开垦心智层”一文研究了“集市”开发模式的社会动力学，指出要想最有效理解这种模式，应参考一下人类学家所称的礼物文化（而不是传统的交换经济），这种文化下，其成员通过送出礼物而竞争社会地位。本文我将首先驳斥一些关于软件产品经济的常见神话，然后将分析延伸到经济学、博弈论和商业模式领域中，并给出一些新的概念性工具，用来理解开源开发者的礼物文化在现实的交换经济中是如何存活的。

为了不分心地沿着这条线索分析，我们需要放弃（或至少暂时忽略）礼物文化这个层面的解释。“开垦心智层”一文假定礼物文化产生的情境是生活用品足够丰富，以至于互相交换方式不再让人觉得很有意思。从心理层面讲，这种解释非常有力，但考虑到大多数开源开发者实际上生活在混合的经济环境下，这种解释就不够充分了。大部分情况下，交换经济失去了吸引力，但并没有失去制约力。只有在物质稀缺的经济环境下也能解释黑客行为，才能让这些行为更好地立足于物质过剩的礼物文化中。

因此，本文会（完全在稀缺经济领域内）思考使开源开发得以生存的合作与交换模式。按此思路分析的同时，我会辅以实例，详细回答这

个实用主义者问题：“我怎样从中赚钱？”该问题产生于人们对软件产品经济学的一般认识，常见而错误，我会首先展示这个问题背后所隐藏的不安心理。

（在论述之前再最后多说几句：本文对开源开发的讨论及支持，并不代表认同“闭源开发在本质上是错误的”，本文并不是对软件知识产权的反对，也不是对无私“共享”的摇旗呐喊。自“大教堂与集市”发布以来，经验表明这种争论完全没有必要，但开源开发社区中少数人仍然热衷于打这些口水仗。对开源开发完整而充分的认识应该建立在其工程上和经济上的结果——质量更好、可靠性更高、成本更低、选择更多。）

4.3　批量制造的错觉

我们首先要注意的是，计算机程序像其他所有类型的工具或生产资料一样，有着两种不同的经济价值：使用价值和销售价值。

程序的使用价值是它作为一个工具、一个生产率倍乘器的经济价值；程序的销售价值是它作为一个可买卖商品的价值。（按照经济学家的专业说法，销售价值是最终产品价值，使用价值是中间产品价值。）

在尝试使用经济学分析软件产品时，大多数人会想当然地运用“工厂模型”，它建立在如下基本假设之上：

· 大多数开发者的薪金由软件销售价值支付。

· 软件销售价值和它的开发投入成比例。

换句话说，人们十分倾向于假设软件具备典型批量商品的价值特点，但这些假设都可以被证伪。

首先，被编写用来出售的代码仅仅是编程冰山的一角。在微机时代早期，人们普遍认同，全世界90%的代码都是在银行和保险公司内部编写的。当然现在不是这样了——其他行业也逐渐成为软件密集型行业，金融行业所占比重相应下降——但我们很快会看到，经验表明大约95%的代码仍然是在机构内部编写的。

这些代码包括大多数MIS系统以及客户化的财务软件或数据库软件，这是每个大中型公司都需要用到的；还包括像设备驱动程序这样的技术专家级代码，而几乎没人通过卖设备驱动程序赚钱，后面我们会再

谈这个；还包括所有类型的嵌入式代码，用于越来越多的微芯片驱动的机器——从机械工具、喷气式客机到汽车、微波炉和烤面包机。

大多数这样的内部代码与它所在的环境密切相关，对它们的重用和复制非常困难。（不论这个环境是商业办公的流程集合，还是联合收割机的燃油喷射系统。）因此，当环境变化时，需要持续不断地保证软件能跟上变化。

这些工作称为“维护”，任何一个软件工程师或系统分析师都会告诉你，支付给程序员报酬的工作中，大多数（超过75%）都是这类。相应地，程序员的工时也大都花费在编写和维护这些没有任何销售价值的内部代码上——读者可以很容易地在报纸“招聘”栏列出的编程工作列表中验证这个事实。

我强烈推荐度读者做一下这个验证实验：浏览你所在地报纸的招聘栏，检验一下它列出的编程、数据处理和软件工程这类涉及软件开发的职位，并对每个这样的工作进行判别，看看它所开发出来的软件是供使用的还是供销售的。

你很快就可以清楚地看到，即便按照最宽泛的“销售”定义，20个职位里也至少有19个完全是由使用价值（也即，中间产品价值）付费的，这就是为什么我们相信行业中只有5%的部分是由销售价值驱动的。注意，即便如此，本文其余部分的分析对这个数据是相对不敏感的，即便这个数是15%甚至是20%，其经济推论结果在本质上也都一样。

当我在技术会议上发言时，我常常会以两个问题作为开始：听众中

有多少人因写软件而获得报酬，又有多少人的报酬是建立在软件的销售价值之上。通常我会看到听众对第一个问题举手如林，而对第二个问题的响应寥寥无几，并且他们会对这个比例感到相当惊讶。

考察一下消费者的实际行为，你会发现，关于软件销售价值取决于其开发或升级成本的理论很容易被推翻。很多商品（在贬值前）的确有这样的关联关系，如食物、汽车和机械工具。甚至有些无形商品的销售价值也和其开发及升级成本紧密相关——如音乐、地图及数据库的复制权。这种商品在其供应商消失后，其销售价值仍然保持甚至还会增加。

相比之下，当一个软件产品供应商退出市场（或仅仅是产品不再延续）后，消费者愿意为其产品支付的价格很快会降低到零，而不管其理论上的使用价值或该类产品的开发成本如何。（要想检验一下这个论断，可以看看你附近软件商店里的打折专柜）。

当软件供应商关张时，零售商的行为很有揭示意义。他们知道一些供应商不知道的东西：事实上，消费者愿意支付的价格不会超过对供应商服务的未来预期价值（“服务”的含义很广，在这里包括完善、升级以及后续产品等等）。

换句话说，软件很大程度上是一个服务行业，虽然长期以来都毫无根据地被错认为是制造行业。

有必要研究一下我们为什么常常倾向于相信这种错觉，原因可能很简单，软件行业中用来销售的那一小部分软件，是唯一做广告的部分。人们通常的心理偏见可能也起了作用，那就是认为制造业比服务业

更“真实”，因为前者生产的东西是看得见摸得着的[1]。另外，一些最明显的、大张旗鼓宣传的产品，其生命周期都非常短暂（比如游戏），这些产品几乎没有持续的服务需求（这是例外，而非一般情况）[2]。

同样值得注意的是，这种错觉鼓励了一种病理性的价格结构，完全脱离了开发成本的实际分解。如果软件生命周期中超过75%的成本（这已经被普遍接受）都花在维护、排错和扩展上，那么常见的价格策略——给软件定以较高的购买价和相对较低甚至为零的维护费——就必然导致很差的服务。

消费者会因此吃亏，虽然软件是一个服务行业，但基于工厂模式的激励机制会阻碍供应商提供足够的服务。如果供应商的收入来自于销售软件，那么大部分的努力就会是制造软件并将它们推销出去；至于帮助台（help desk），它们不是盈利中心，将会成为最低效员工的发配地，并只能得到仅供维持客户数量不低于临界值的资源。

情况会变得更糟。实际使用就意味着会有人打服务电话，而提供服务会影响利润率，除非你对服务也收费。在开源世界里，你寻找的是最大可能的用户群，以便获得最大限度的反馈和最有活力的可能的二级市场。在闭源环境中，你寻求的是尽可能多的购买者和尽可能少的实际使用者。所以，工厂模式逻辑会强烈鼓励供应商生产出“存架软件”——市场销量很好但没有实际价值的软件。

事情的另一面是，大多数信奉工厂模式的供应商会在长期运转中失败。以固定的销售价格支付永无期限的服务费用，只有在一种情况下才

行得通：市场快速扩张以至于能用昨天的销售和明天的收入来覆盖软件支持费用和生命周期成本，一旦市场成熟，销售放慢，为削减开支，大多数供应商将不得不放弃维护和服务，使软件沦为“孤儿产品”。[3]

不论这种做法是显式（中止产品）还是隐式（使顾客难以得到服务）的，都会起到把消费者赶往竞争者那里的效果——因为它破坏了产品的取决于服务的预期未来价值。供应商可以采取把bug修复版本包装成新产品并贴上新价格这种办法，但这是避开困境的短视行为。从长远看，唯一能避开困境的办法是没有竞争者——也即在市场上实现有效的垄断。到最后，只有一家供应商。

我们确实已经多次看到“支持匮乏”这种失败模式在市场环境中甚至干掉了排名第二的强有力竞争者。（那些调查过PC操作系统、字处理软件、会计程序或一般商业软件历史的人，会很清楚这种模式。）这种由工厂模型带来的非正常激励，导致了“赢家通吃”的市场模式，并最终导致消费者（即便是赢家的消费者）的损失。

如果不用工厂模型，用什么？要想有效（不管是非正式意义上还是经济学意义上）调整软件生命周期的真实成本结构，我们需要一个建立在服务合约、按期订购以及在供应商与消费者之间价值持续交换基础之上的价格结构。ERP（企业资源规划）系统这种最大商业软件产品的价格结构正是如此，由于其开发成本过于庞大，固定采购价格不可能完全覆盖，像Baan和Peoplesoft这样的公司实际上是通过售后咨询赚钱。在追求效能为前提的自由市场中，我们可以预测，在软件行业成熟之后，

这将成为大多数厂商最终遵循的价格结构。

以上分析让我们明白，开源软件不仅在技术上，而且也在经济上挑战着现有秩序。将软件“免费”所带来的效果，看来会强制我们进入以服务费为主导的世界，并让我们清楚了解，闭源软件一直以来依靠软件销售价值的做法是多么脆弱。

这个转变并不像表面看起来那么令人难以接受。当盒装软件（尤其是游戏、操作系统和受欢迎的生产工具）的盗版拷贝能够轻而易举搞到手的时候，从消费者角度看，只有在得到如下好处时，才值得为专有软件付钱：供应商服务、纸质手册，或者一种有道德的感觉。所谓“免费”软件的商业发布版，也正是按照这种方式对消费者收取费用——唯一区别是这些供应商不会愚弄消费者，不会让他们认为仅仅那些二进制比特就有着不可或缺的价值。

“免费”这个词还容易在其他方面造成误解。为降低商品成本，投在维护上（包括相应的人和基础设施）的钱，往往是增加而不是减少。当汽车价格降低时，汽车修理工的需求就会增加——这就是即便目前靠销售价值养活的那5%的程序员也不会在开源世界中难以过活的原因。在这个转变过程中，有损失的不会是程序员，而是那些不顾经济规律把赌注放在闭源策略上的投资者。

4.4 “信息要免费”的神话

“信息要免费”（Information Want To Be Free）是一个与“工厂模型”对立但同样错误的神话，思考开源软件经济学的人常为此困惑。它常体现为这样一种主张：数字信息的复制成本接近于零，所以其结算价格应该为零（或表述为：在充满复制者的市场中，其价格不得不降为零）。

有些类型的信息确实需要免费，不准确地讲，当越多的人能够接触到它，它的价值就会越高——技术标准文档就是一个很好的例子。但如果认为所有信息都该免费，那可经不起推敲，考虑一下那些使人们能够获取排他性好处的信息——一张藏宝图、一个瑞士银行账号，或一个可用来索取服务的信息如计算机账号密码。即使这些信息的复制成本可以为零，该信息所能索取对象的成本也不会为零。因此，被索取对象非零成本的特性会继承到该信息上。

我们提及这个神话，主要是为了说明它和开源经济的效用参数几乎毫无关系，就像我们后面将会看到的，即便假设软件确实有着（非零的）制造业产品价值结构，这个论断仍然成立，所以我们没有必要考虑软件是否应该免费这个问题。

4.5 反公地模型

在对当前流行的模型投以怀疑目光的同时，我们看看是否能够建立起另一个模型——一个在经济学上的可靠模型，用来阐述开源合作的可持续性。

它要能经得起几个不同层面的检验。一方面，我们需要解释开源项目贡献者的个体行为；另一方面，需要理解那些支撑Linux或Apache这类开源项目合作的经济力量。

再一次，我们需要先推翻一个妨碍理解的流传很广的民间模型。因为在每次试图对合作行为做出解释时，Garret Hardin的“公地悲剧”理论都会投来一片阴影。

在这个著名的理论中，Hardin假想在某个村子里有一片公共的草地，每个村民都可以在这片草地上放牧，长期放牧使得土地退化、草皮损坏、到处都是泥坑，而草皮的恢复很慢，如果没有一个一致同意的（或者强制的）放牧权分配策略，就很难抑制过度放牧，出于自身利益考虑，每个人都会尽可能多和尽可能快地放牧自家牛羊，以便在公地退化成泥沼之前，从中获取最大价值。

大多数人有着和此类似的本能的合作行为模型。公地悲剧事实上来自于两个互相联系的问题，一是过度使用，二是供应不足。在需求端，公地鼓励“竞次”行为 [2]而导致过度使用——也即经济学家所称的公益拥挤问题。在供给端，公地奖励了“搭便车”行为 [3]——它打消或抑制了个

体在开发新牧地上的投资意愿。

公地悲剧预言了三个可能的结果。一是公地退化为泥沼，二是少数有强制力的成员代表全村实施分配策略，三是将公地分割给村民，让每个村民自行保护（如围起篱笆）和管理自己那块草地的可持续性。

当把这个模型下意识地套用在开源合作上时，人们预期开源合作是不稳定的，而且有着很短暂的半衰期。由于显然没有什么办法能通过互联网强制分配程序员的工作时间，使用“公地悲剧”模型将直接得出预测：就像公地将会被分割那样，软件将会被分为很多块并闭源起来，回馈在公共代码池上的工作量会迅速减少。

经验表明，事实上的趋势与此相反，开源开发在范围和数量上的趋势，可以从每天在Metalab和SourceForge（Linux源代码站点的领头雁）上的提交数或每天在freshmeat.net（一个致力于推介新软件发布的站点）上的通告数测算出来，它们都有着稳定而快速的增长。“公地悲剧”模型一定在某些关键的地方和实际不符。

一部分原因当然是因为使用软件并不会降低软件的价值。事实上，对于开源软件来说，使用得越广泛，就越可能增加其价值，因为用户会提供软件bug修复和功能补充（代码补丁）。反公地模型中，在软件上“放牧”越多，“草”反而长得越好。

所以，开源软件的公共利益不会因为过度使用而恶化，但这只考虑了“公地悲剧”中的公益拥挤问题，还没有解释为什么开源不会遭遇供应不足的问题。为什么明知道开源社区中普遍存在搭便车行为，人们也不

会去等别人去做自己需要的东西，而且（自己）也不在意向公地做出贡献？

部分原因是人们需要的不仅仅是解决方案，他们还需要问题的及时解决。对于某个给定需求，几乎不能预期别人会在什么时候搞定它。对任何一个潜在贡献者而言，如果他从bug修复或功能增加中能获得足够的回报，他就会投入其中（即便所有人都搭便车也没关系）。

另一部分原因在于，对于公共代码的微小补丁，其公认的市场价值很难收取。假设我写了一段代码，可以解决掉某个烦人的bug，并且假设有很多人愿意为这个补丁付钱，可我怎样向这些人收费？传统的支付系统有着很高的管理成本，这使得各种合理的小额支付成为一个实实在在的问题。

还不仅仅是难以收费的问题，可能更切中要害的是，通常情况下你很难定价。让我们做一个思想实验，假设互联网上已经有了理论上很完美的小额支付系统——安全，普及，零管理成本，现在，你写了一个叫做“Linux内核若干修补”的补丁包，你如何定价？对于一个可能的购买者，在还没有拿到这个补丁的时候，他怎么知道该付多少钱？

这个问题很像是F.A.Hayek所提“计算问题”的翻版——它需要一个超能力（superbeing）存在：既能评估补丁的实用价值，又能可信地设定其价格以促成交易。

遗憾的是，我们并没有这种超能力，补丁的作者J.Random（J.Random用来泛指某位黑客——译者注）只有两个选择：坐

等在补丁上，或者将其免费投入到公共代码池中。

坐等在补丁上不会获取任何收益。事实上，它会招致未来的成本——在系统每次新发布时，都需要将补丁重新合并到源代码中。从这种选择中获取的收益，其实是负值（并且会因开源项目发布的快节奏特点而倍增）。

如果积极一点，将补丁贡献出来，维护补丁的日常开销会转移给源代码所有者和项目团队中的其他人，而且这个补丁可能会在未来某个时候被他人改进，由于并不是非要作者本人维护这个补丁，他可以有更多时间满足自己对软件更多的个性化需求。这些好处不仅体现在补丁上，也同样适用于整个软件包。

将补丁投入到公共代码池中可能不会有什么收获，但会鼓励别人参与进来，并在某个时候解决J.Random碰到的问题。从表面上看，这是利他的，但从博弈论意义上讲，这是最大化的利己。

分析这类合作时需要注意的是，搭便车问题（由于缺乏金钱或等价补偿而带来的供应不足问题）并不是随最终用户增多而凸显的唯一问题（详见书后注释1）。开源项目的复杂性和沟通成本基本上完全是参与开发人数的函数，大多数从来不看源码的终端用户实际上并不会带来什么成本，但会增加项目邮件列表上愚蠢问题的比例，好在可以通过维护FAQ（常见问题）列表较为轻松地解决这个问题，而且通常大可不必理会那些明显没有读过FAQ的提问者（这些已经是通行做法）。

开源软件中真正的搭便车问题，其实更多体现在提交补丁时的磨合

成本。一个潜在的贡献者，在黑客文化的声誉游戏（见“开垦心智层”一文）中，几乎没有什么赌注，在缺乏金钱补偿的情况下，会觉得“不值得提交这个修复，因为我不得不整理出补丁，写修改记录，还要签署FSF版权转让书……所以，贡献者数目以及（由此而导致的）项目成功程度，和项目对贡献者所设限制条件多少是强烈负相关的。这种磨合成本可能是政治上的，也可能是机制上的。总之，我认为这解释了为什么宽松的、无组织的Linux文化，比起相对严格的、有组织的、集中式管理的BSD文化，能吸引更多的合作能量（远不在一个数量级上）。这也可以解释为什么FSF在Linux兴起之后变得不再那么重要。

目前看来这些分析都没错，但它们只是对黑客如何对待补丁（在写好补丁之后）的一个事后解释，我们需要对问题的另一半做出经济解释，即J.Random为什么要写免费补丁，而不是通过闭源程序获取销售价值？是怎样的商业模式，造就了能让开源开发蓬勃发展的生态环境？

4.6 闭源的理由

在对开源商业模型进行分类之前，我们应该先研究一下闭源有什么好处，闭源真正想保护的是什么？

假如你雇人给你的企业写了一个专业的财会软件，选择闭源不会比开源好到哪里，如果你希望闭源，唯一合理的原因是你想把这个软件卖给别人，或者防止竞争者使用它。

答案很明显，你在保护销售价值，但95%的软件写出来是供内部使用的，在没有销售价值可言的情形下，从闭源中还能得到什么好处？

需要稍微分析一下的是第二种情况（保护竞争优势），假设你把这个财会软件开源了，它变得很受欢迎，在社区的帮助下它也变得更好了。你的竞争对手也开始使用它，竞争对手没有支付开发成本却获得好处，而且还影响到你的业务，这是不是一个反对开源的理由？

也许是，也许不是。真正应该考虑的问题是：你从分散开发负担中获取的益处是否超过了因（“搭便车”行为导致的）竞争加剧而带来的损失，一些人往往在这个权衡中失算：(a)忽略了社区开发带来的功能改进。(b)不把已经支出的开发成本当做沉没成本。根据假设，不管怎样，你都是要付出开发成本的，所以把它归入开放源码（如果你这样认为）的成本是不对的。

另一个经常被提及的担心是，将某些特别的财会功能开源，会不会导致商业机密方案的泄露？其实这不关开源闭源的事，这是糟糕设计带

来的问题。财会软件如果编写得当，商业知识是不会在代码中体现的，它应该由一个模型（schema）或描述语言表达，然后由财会引擎执行实现（作为很相近的一个例子，考虑数据模型将业务知识和数据库引擎相分离的做法），这种功能上的分离使你不但可以保护住王冠上的宝石（即你的商业模型），还能从开放引擎中获得最大收益。

还有其他一些理由可以说明闭源完全是不明智的。你也许被一些谬论误导，觉得闭源能使你的商业系统更安全地防范骇客入侵，如果是这样，我建议你立刻和密码工作者做一次彻底的交谈。比起用闭源来保障安全性，真正专业而审慎的人知道什么才是更好的做法，他们已经从惨痛教训中学到太多了。安全性是可靠性的一部分，只有在算法及其实现经过彻底的同行评审之后，才有可能被认为是安全的。

4.7 “使用价值”资助模型

使用价值和销售价值之间的差别，让我们注意到这样一个关键事实：在从闭源转向开源的过程中，受到威胁的仅仅是销售价值，而非使用价值。

如果使用价值真的是软件开发的主要驱动力，并且开源开发真的比闭源更有效也更快捷（在“大教堂与集市”一文中已经论证了），我们应该能够发现一种仅靠使用价值就能持续资助开源开发的模式。

事实上这不难找到，至少存在两种重要的开源项目模式，使其全职开发者的薪水直接由使用价值支付。

APAHCE案例：成本分摊

假设你为一个公司工作，该公司有一项关键业务，需要高吞吐量和高可靠的Web服务器。它上面跑的也许是电子商务，也许是一个高可视性的销售广告媒体系统，也许是一个门户网站，你需要24/7的不间断运行，你需要速度和定制性。

如何做到这些？有三种基本策略可供选择：

买一个专有的Web服务器。

这种情况下，你是在赌，赌卖家的研发计划能匹配你的计划，赌卖家的技术能力能实现你的需求。即便这些都没问题，这个产品也很可能是缺乏定制性的，你只能通过供应商给你的接口来修改它。在Netcraft每月的报告中可以看到，这种专有路线已经不流行了，并且越来越不受

欢迎。

自己做一个。

做一个自己的Web服务器不是不可能的，Web服务器并不很复杂，肯定比浏览器要简单些，而专门研制的Web服务器可能会很轻便也很能干。走这条路，你可以获得精准的功能和所需的定制性，尽管你需要为此付出开发时间。而你的公司，可能会在你退休或离职后，发现它有问题。

加入Apache团队

Apache服务器是由一个通过互联网连接起来的团队（成员多为网站管理员）做出来的，他们明白，与其很多人都搞类似的开发，不如将自己的工作投入到公共代码中。通过这种方式，他们不仅能获得“自己做一个Web服务器”的好处，还会因为大规模的同行评审而获得强大的排错效果。

选择Apache的优势很强，究竟有多强，我们可以从Netcraft每月的统计报告中判断，自Apache面世以来，其市场占有率稳步上升，专有Web服务器的占有率则相应下降。2000年11月，Apache及其变种取得了60%的市场份额（http://www.netcraft.com/survey/），这还是在Apache没有法定所有者、没有宣传、背后也没有签约服务机构的情况下获取的。

Apache案例给出了开源开发的一种协作资助模式，存在竞争关系的软件用户发现，该模式使得各方都能获益，因为只有这样，才能以更低的成本提供更好的产品。

Cisco案例：风险分担

若干年前，Cisco（一家网络设备制造商）两名程序员被指派编写一个用于Cisco企业网的分布式假脱机打印（print-spooling）系统。这个活可不容易，它除了要求支持任意用户A能在任意打印机B上(它可能在隔壁屋里或是在千里之外)打印东西外，还要求在B缺纸或墨粉不足时，确保打印任务能被重新路由到B附近的一个候补打印机上，而且还能将这种问题报告给打印机管理员。

这二人在标准的UNIX假脱机打印软件上做了一系列巧妙的改动（http://www.tpp.org/CiscoPrint/），然后编些脚本封装一下，就把工作搞定了。不过他们很快意识到有个问题。

问题就是，这二人不可能永远呆在Cisco，他们迟早都会离开，软件将会失去维护并慢慢腐烂（也即逐渐不能跟上现实的变化）。没有开发者愿意看到自己作品的结局变成这样，二人觉得，Cisco已经为这个方案支付了成本，公司当然期望系统的使用期限超过他俩的雇佣期限。

于是，他们跑到经理那里，强烈建议经理授权将这个假脱机打印系统开源。理由是，该软件本身没有销售价值，Cisco不会因此损失什么，但却会获益良多。通过鼓励社区（由遍布在多个公司的用户和合作开发者组成）成长，Cisco能对冲原始开发者离开的损失。

Cisco案例表明：开源不仅可以降低成本，还能分散和减轻风险。所有参与方都发现，源码的开放与合作社区（由多个独立的收入来源资助）的存在，提供了一种故障保险（fail-safe）机制，其本身的经济价

值足以吸引资金支持。

4.8　为什么销售价值问题多多

源码开放使得从软件中直接获取销售价值变得更加困难，这个困难并不是技术上的，因为源代码并不比二进制更难或更容易被拷贝，而版权和许可证的法律效力，也不会让开源比闭源更难获取销售价值。

困难存在于开源开发社会契约的内在特点。有三个相辅相成的原因，使得主流的开源许可证不允许对对开源软件使用、再发布和修改施加限制，从而影响直接销售收入的获取。要理解这些原因，我们必须研究许可证演变所处的社会语境，也即互联网黑客文化（http://www.tuxedo.org/~esr/faqs/hacker-howto.html）。

尽管黑客圈外仍然广泛流传着“黑客敌视市场”的谬论，但这和真正原因无关，确实有少数黑客很反感盈利动机，但从黑客社区和营利性Linux软件包发行商(packagers)如Red Hat、SuSE和Caldera的普遍合作意愿不难看出，大多数黑客乐于和商业世界（是他们向最终用户提供服务）合作。黑客对直接收费类型许可证心存不满的真正原因是什么？这个问题微妙而有趣。

第一个原因与“对等性”有关，大多数开源开发者并不反对别人利用他们的礼物获利，只是要求不能有任何人（代码创始人可能会例外）站在一个特权地位上牟利。J.Random愿意让Fubarco（Fubarco泛指某家公司——译者注）卖自己软件（或补丁）的前提是他本人也有这样的权利。

第二个原因则与“非有意后果”有关。黑客已经观察到，那些对商业使用或销售进行限制并收费（这是最常见的征费方式，乍看上去并无不妥）的许可证有着令人扫兴的效果。特别是这条规定给某些活动（如将开源软件系列发布在便宜的CD-ROM上）笼上了一层法律阴影，而这些活动正是我们非常愿意鼓励的事。更普遍地讲，如果对软件的使用/销售/修改/发布(以及其他在许可证中描述的复杂情况)加以限制，会使人们总是小心翼翼防范那些不确定的和潜在的法律风险（当人们接触的软件包越多，这个问题就越严重）。这个结果是有害的，因此，在强大的社会压力下，许可证将会变得越来越简单和无限制。

最后也是最关键的一个原因，与保持同侪评价这种礼物文化动力（“开垦心智层”一文中所描述的）相关。如果许可证被设计用来保护知识产权或直接获取销售价值，它通常就会在法律上使项目不可能产生分支，举例来说，Sun用于Jini和Java的所谓“社区源码”许可证就导致这种情况。虽然黑客们不赞成分支（其原因在“开垦心智层”中已经详尽分析了），但是会把另辟分支当做“最后一招”，当维护者不能胜任或变节（比如走向更封闭）的情况下[4]，人们可以使出这关键的“最后一招”。

黑客社区会在“对等性”原因上有所让步，它能容忍Netscape Public License（NPL）赋予代码创始人以特权（具体来说，NPL中规定创始人可以使用Mozilla开源代码派生含有闭源的产品）。黑客较少在“非有意后果”原因上让步，至于在许可证中禁止产生分支，黑客们是绝对不答应的，这也是Sun关于Java和Jini的社区源码许可证方案被开源社区普遍

拒绝的原因。

（这里有必要再次重申，黑客社区里没人愿意看到项目分裂成互相竞争的多条开发线，正如我在“开垦心智层”一文中所提到的，出于良好的愿望，人们对分支非常反感。就像没人愿意身处警戒线上、法庭里或者枪战中，分支的权力就好比是罢工的权力、起诉的权力或是拥有枪支的权力——不到万不得已，你并不想使用这些权力，但如果有人试图夺去这些权力，那一定是非常危险的信号。）

以上原因解释了“开源定义”（Open Source Definition）的某些条款，这些条款表达了黑客社区的共识，也是标准许可证（如GPL、BSD许可证、MIT许可证和艺术许可证（Artistic License））的关键特性。正是这些条款，导致了（尽管不是有意的）开源社区很难直接获取销售价值。

4.9　非直接的销售价值模型

尽管如此，还是可以通过创造软件服务市场获取一些非直接的销售价值。目前有5种已知的和2种尚不确定的模型（未来还会开发出更多的模型）。

占领市场

在这种模型中，你可以利用开源软件建立或维护专有软件（能产生直接收入）的市场地位。最常见的形式是，将客户端软件开源，用来帮助服务器端软件的销售，或帮助门户站点获得订阅及广告。

网景通信公司1998年开放Mozilla浏览器的做法，就使用了这种策略，浏览器收入占据他们总收入的13%，在微软推出IE后，这个比例不断下降，IE强大的市场营销（其备受质疑的捆绑行为很快成为反垄断诉讼的焦点问题）使其迅速吞占了网景的浏览器市场份额，这不免让网景担心，微软将会垄断浏览器市场，然后通过对HTML和HTTP的事实上控制，将网景的服务器端逐出市场。

在网景浏览器仍然广受欢迎时将其开源，网景有效地反击了微软垄断浏览器市场的可能。他们期望开源合作可以加速浏览器的开发和排错，并借此降低微软IE的追赶速度，防范微软对HTML标准的排他性操纵。

这个策略起作用了，1998年11月，网景真的从IE那里夺回了一些市场份额。在1999年初网景被AOL收购时，Mozilla项目带来的竞争优势非

常明显，所以AOL的首批公开承诺之一就是继续支持Mozilla项目，尽管当时它仍处于alpha版本阶段。

硬件糖霜

这种模式适用于硬件制造商（这里的硬件包括网卡、外围设备乃至整个计算机系统）。市场压力迫使硬件公司编写和维护软件（设备驱动程序、配置工具乃至整个操作系统），但软件本身不是盈利中心，它是一项开销——并且常常是一项很实在的开销。

这种情况下，开源是最好的选择，因为并没有收入流可以损失，也没有什么负面效果。供应商可以获得快速膨胀的开发者队伍，对客户需求会有更快和更灵活的响应，并因同行评审获得更好的可靠性。它会被免费移植到其他环境下，可能还会获得客户忠诚度的提高，因为客户的技术团队会投入更多的时间根据其需要改进代码。

有几个反对意见（特别是针对硬件驱动程序的开源）常被供应商提及，为了不把这些反对意见和更一般的情况混杂起来，我会在后面专门讨论这个话题（参见“后记：为什么驱动程序闭源会让制造商坐失良机”一节）

硬件糖霜模型很好体现了开放源码的“防患于未然”（future-proofing）效果，硬件产品生产和支持的生命期都是有限的，在此之后，客户就只能靠自己了。但如果客户能够接触到驱动程序源码而且还能给程序打补丁，他们可能会很高兴成为回头客。

苹果公司于1993年3月中旬将“Darwin”（MAC OS X服务器操作系统

的核心）开源的决策，是采纳硬件糖霜模型的一个经典事例。

送配方，开餐馆

在这种模型中，开源软件不是为闭源软件而是为服务占领市场。

（我常把这种模型称为“送剃须刀，卖刀片”，虽然这个比喻不是很贴切。）

Cygnus Solutions最先使用这个模型，大概这是开源的第一次商业行为（1989）。当时，GNU工具集提供了一个可用于多种机器的通用的开发环境，但在不同的平台下，每个工具的配置过程都不一样，并且要打上不同的补丁包。Cygnus改良了GNU工具集并写了一个“配置”脚本来统一构建过程（也即配方），然后，他们出售这个版本的GNU工具集以及与之绑定的二进制软件及支持服务（也即餐馆）。遵从GPL要求，他们允许客户自由使用、发布和修改这些软件，但若使用支持服务的用户超出合同所承诺的数量，服务合同将会中止或需要支付更高的费用（也即，沙拉吧是不免费的）。

这同样是Red Hat和其他Linux发行商所采用的做法。他们真正销售的东西不是软件，不是那些比特，而是他们所提供的附加价值：他们通过集成和测试，向客户提供一个不断发展的操作系统，保证它是商业级的（要是再低调点就好了），并可以和该品牌其他操作系统向后兼容。他们还提供免费的安装支持和可选的持续支持合约。

源码开放会产生极为强大的市场效果，特别是对那些一开始就定位于提供服务的公司，一个非常有启发意义的例子是“Digital Creations”，

它创立于1998年，是一个网站设计公司，专长是复杂数据库和交易类网站。他们的主要工具——也是该公司知识产权王冠上的宝石——是一个对象发布环境，在经历若干个名字和版本后，该工具现在叫Zope。

当Digital Creations寻求风险投资（VC）的时候，他们找来风险投资人评估他们的市场定位、人和工具，VC建议Digital Creations把Zope开源。

从常规的软件工业标准来看，这绝对是疯狂的举动。商学院的传统理念认为，像Zope这种核心知识产权是公司王冠上的宝石，无论如何都不应该公开出来。但VC洞察到两点：一是Zope的真正核心资产是公司员工的智慧和技能，二是Zope开源后（而不是作为一个秘密工具）会产生更多的市场价值。

为了看得更清楚，我们可以对比一下两种场景：在传统的场景下，Zope仍然是Digital Creations的秘密武器，假设它仍然非常有效，公司能够在短时间内交付较高质量的产出——但没人知道他们是怎么做的。它较容易让客户满意，但是很难培养出基于该软件的客户群。

而VC认为，开源的Zope将会给Digital Creations的真正财产——即公司的员工——带来非常重要的广告效应，VC预期客户在评估Zope时，会认为雇佣专家要比在公司内部积累Zope技能更有效率。

Zope的一位高管已经非常公开地表示，开源策略让他们“打开了很多扇其他方法无法打开的门”。潜在客户的确响应了这个逻辑——Digital Creations则相应地繁荣起来。

另一个例子是e-smith公司（http://www.e-smith.net/）。该公司出售他们“交钥匙”型互联网服务器软件（一种开源的客户化Linux）的支持服务，一位高管在描述他们软件可以被免费下载并传播的情况时说：“大多数公司会认为这是盗版，但我们认为这是自由市场的行为”（http://www.globetechnology.com/gam/News/19990625/BAND.html）。

附属物策略

在这种模型下，人们出售开源软件的附属物，如低端市场上的杯子或T恤，高端市场上的专业书籍或文章。

附属物策略的一个很好例子是O'Reilly&Associates公司，它是很多优秀开源软件参考书的出版商。为提高在市场上的声誉，O'Reilly实际上雇佣和支持了一些著名的开源黑客（如Larry Wall和Brian Behlendorf）。

当下收费，未来免费

这种模型以封闭许可证的形式发布软件的二进制及源码，但其中的封闭条款有过期时间。比如，你可以在许可证中允许自由发布、禁止免费的商业使用，并保证在发布一年后或者供应商关张后，软件将遵循GPL条款。

这种模型的好处在于，它使顾客确信产品是可以客户化的——因为他们有源码，而且产品是“始终可用”的，因为当原创公司消亡后，开源社区会接管起来。

销售价格和销售量取决于客户预期。比起在发布时采用排他性闭源许可证，使用这种模型会享有更多的收入。而且，由于许可证过期后代码将遵循GPL，因而会获得严格的同行审查、bug修补和细节功能增加，这会减少原创者75%的维护负担。

这种模型已经被Aladdin Enterprises公司成功实施，它做出了流行的Ghostscript程序（一个PostScript解释器，可以把PostScript翻译成多种打印机语言）。

这种模型的主要缺点在于，在产品早期，封闭条款往往限制了同行评审和参与，而那时是最需要这些的时候。

软件免费，品牌收费

这是一个探索性的商业模型。你可以开源一项软件技术，保留其测试套件或兼容性标准，然后卖品牌认证，如果某公司的产品通过认证，则表明他们对该技术的实现和其他拥有该品牌认证的产品是兼容的。

（这是Sun公司应该对待Java和Jini的方式。）

更新：在2000年7月，Sun声明它将开源Star Office，对于在此代码基础上开发的经Sun认证的系列产品，可向Sun购买对Star Office品牌的使用。

软件免费，内容收费

这是另一种探索中的商业模式。想象一个类似股票报价订阅的服务，其价值既不在客户端也不在服务器端，而在于提供客观可信的信息。所以，你可以开源软件而销售内容订阅。当黑客们把客户端移植到

新平台或以各种方式做了改进时，你的市场也自然而然地扩展了。

（所以AOL应该开源其客户端软件。）

4.10　何时开放，何时关闭

在了解了这几种支持开源软件开发的商业模型之后，我们可以研究一下：何时开源及何时闭源会产生经济价值。首先必须弄清楚不同策略的收益。

收益是什么？

走闭源这条路，你能够通过秘密比特赚钱，但也使得真正独立的同行评审变得不可行。开源这条路为独立同行评审创造了条件，但你却不能从比特中赚钱。

秘密比特带来的收益很好理解，传统软件商业模式就构建于其上。而独立同行评审的价值直到最近才被人们认识到，Linux操作系统给我们上了一课——其实我们早该在多年前就从互联网核心软件及其分支中了解到——开源的同行评审是得到高可靠和高质量软件的唯一的可扩展方法。

在一个竞争性市场上，客户寻求的是高可靠和高质量的软件，走开源之路并学会如何通过服务、增值和附属市场获取收入流的软件制造者，自然会赢得客户的回报。这正是导致Linux取得令人震惊成功的原因，1996年的Linux还什么都不是，但到2000年中，它已经成为商业服务器市场的第二大操作系统。IDC在1999年初预测，到2003年，Linux的成长速度将会超过所有其他操作系统的总和，目前看来这个预测仍然有效。

另一个几乎同等重要的收益是，开源是标准传播和市场创建的有效途径。Internet的戏剧性成长，在很大程度上要归功于没人拥有TCP/IP标准，也就是说，没有特权能够控制Internet的核心协议。

TCP/IP和Linux的成功所带来的网络效应[4]已经非常明显，并且它非常好地解决了信任和对等问题——对于一个可能的参与者，如果他能看到大家共享的基础架构是怎么工作的（包括所有细节），自然会更信任它一些。他也更愿意选择一个所有参与者都有着对等权力的基础架构，而不是由特权个体掌控并收费的基础架构。

然而，网络效应带来的对等问题，对消费者来说倒不一定那么重要。如果有任何质量上可以接受的开源选择，没有哪个软件消费者在理性分析后仍选择把自己绑定在由某个供应商控制的垄断之上。软件对消费者的业务越关键，这一点就越明显——即软件越重要，消费者就越无法容忍被外人控制。

另一方面，经济学家知道，信息不对称会给市场带来负面影响。如果靠特权信息收费能够比努力生产好产品更有利可图，高质量的产品就会被逐出市场。不透明是质量的敌人（secrecy is the enemy of quality），软件业如此，其他行业也如此。

最后，消费者的另一个重要收益与信任相关，即开源拥有的“防患于未然”效应：如果源码是开放的，消费者在供应商破产后仍然能有所依靠。这对“硬件糖霜”模型来说尤其重要，因为硬件往往只有短暂的生命周期，但此效应是普适的，所有类型的开源软件都会因此而增加内在价

值。

它们如何相互作用

如果从秘密比特中能获取比开源更多的收入，从经济意义上讲就应该闭源；但当从开源中能获取更多收益时，那就应该转向开源。

看上去这很简单，但当我们意识到开源收益比闭源收益更难计算和预测时，就不是那么回事了，大多数情况下，开源收益被远远低估而不是高估。主流商业世界直到1998年Mozilla开源后，才开始反思他们对开源的“想当然”认识，那之前，开源收益被错误而普通地臆断为零。

那如何评估从开源中获得的好处？这是个很难回答的问题，但我们可以像对待任何其他预言性问题一样，从考察一些成功或失败的开源个案开始，先试着归纳出一个至少能给我们以定性感觉的模型，也即在什么情况下，对追求最大回报的投资者或生意人来说，开源是能产生净收益的，然后我们可以再回到数据之中并试着细化这个模型。

根据“大教堂与集市”一文的分析，开源获取高收益的条件大约有如下几种：(a)当可靠性/稳定性/可扩展性至关重要时，(b)没有其他方法比独立同行评审能更便捷易行地验证设计和实现正确性时（多数稍具规模的程序都适用这条）。

当软件对消费者越来越重要时，消费者会在理性上希望避开垄断供应者，这导致他们对开源的兴趣变大（开源供应商的市场竞争力会因此增强），所以，另一个判断标准是：(c)当软件成为对业务起关键作用的资产（比如存在于很多企业的MIS部门中）时。

在应用领域，我们在前面已经看到，开源的基础架构会产生信任和对等，随着时间推移，它会吸引更多客户，并在和闭源基础架构的竞争中胜出。一个较小但快速扩展的市场往往比较大但封闭而停滞的市场要好，一样地，对于基础软件，让自己无处不在的开源玩法要比靠知识产权收费的闭源玩法，能获得更高的长期收益。

事实上，潜在客户会根据供应商的商业策略来分析未来可能的问题，而且他们不愿接受供应商的垄断，这给闭源带来了更强的局限性，如果没有压倒性的市场控制力，供应商要么选择开源（让它普及起来），要么通过闭源直接收费——但不能都选。（其他领域也会有类似问题，比如在电子市场上，客户通常拒绝购买单一货源的设计。）说得再保守一些：在网络效应（正的网络外部性）主导的地方，开源很可能是正确的。

通过观察我们能总结出：(d)当创建或运转一个公共计算或通信基础架构时，开源总是能够非常成功地比闭源获取更多的回报。

最后，我们注意到，提供独特或高度差异化服务的供应商更担心其他竞争者拷贝他们的方法，而关键算法和知识库已经公开化时就不会这样。所以，(e)当关键方法（或能实现同等功能的方法）属于公共知识时，开源更可能胜出。

互联网核心软件Apache以及标准UNIX API的Linux实现是满足这五个条件的最典型案例，这类市场必然朝着开源的方向演进，一个很好的例证是，到20世纪90年代中期，那些试图使用封闭协议（如DECNET、

XNS、IPX等）占领市场的尝试，在历经15年的努力后，无一例外的都失败了，数据网络架构最终纷纷采用了TCP/IP。

从另一方面看，这样的公司是最没有必要开源的：它拥有自己独特的能创造价值的软件技术（完全不满足(e)），它对故障不是很敏感(a)，它有其他办法（不通过独立的同行评审）验证软件的正确性(b)，它不是关键业务(c)，也不会因为网络效应或人们普遍使用而获得价值上的实质增长(d)。

下面是一个极端的例子，一家为锯机编写软件的公司在1999年初向我咨询："我们是否应该开放源码？"，这是一个计算如何切割原木以获得最大板材的软件，我的结论是"不"。他们唯一接近满足的条件是(c)，但即便软件出点问题，一个有经验的操作员也可以手工算出切割模式。

注意，如果这个切割模式计算软件是由锯机设备制造商写的，我的答案可就不一样了，这种情况下，将源码开放会增加他们所售硬件的价值。另外要注意的是，如果该类软件存在开源产品（也许正是某个锯机设备制造商写的），那闭源产品就很难竞争过它——价格是一方面原因，更重要的是客户会因为开源的可定制性和其他优势动心。

还有一点要注意，某个产品或技术对这些条件上的满足会随时间变化而变化，下面的案例分析中我们将看到这点。

总而言之，如果满足下面这些条件，就该考虑把源码开放：

1.可靠性/稳定性/可扩展性非常重要。

2.除了独立的同行评审，没有其他便捷易行的方法验证设计和实现

的正确性。

3.该软件对客户的业务非常关键。

4.该软件创建或运转一个公共计算或通信基础架构。

5.关键方法（或能实现同等功能的方法）属于公共知识。

DOOM案例分析

Doom是id软件公司卖的最好的游戏软件，其历史告诉我们，市场压力和产品演化是如何对闭源产品的收益产生关键性的影响。

Doom在1993年末第一次发布时，它的第一人称视角和实时动画是极为独特的（和条件e对立）。不仅因为它那令人瞠目的视觉效果（远远超过了id公司上一个游戏Wolfen-stein 3D（德军总部3D）的平面动画效果），而且在其发布后数月内没人知道他们是怎样在当时性能较低的处理器上做到的。这些秘密比特确实能换回非常可观的收益，而将代码开源的潜在收益并不大，作为一个单人游戏，这个软件(a)即便有故障，其代价也很低，(b)它不是非常难以验证，(c)也不是哪个消费者的关键业务，而且(d)也不存在网络效应。所以将Doom闭源在经济上是合理的。

然而，Doom的潜在竞争者并没有闲着，他们发明了可与之匹敌的动画技术，其他“第一人称射击”游戏诸如Duke Nukem（毁灭公爵）开始出现，随着这些游戏对Doom市场份额的吞食，秘密比特的收益开始下降。

另一方面，扩大市场份额的努力带来了新的技术挑战——更好的可靠性、更多的游戏特性、更大的用户群和更多的支持平台。随着多人对

战模式（deathmatch）和Doom游戏服务的出现，市场开始出现网络效应。实现这些需要大量的程序员工时，虽然id公司更希望把这些人力花在下一个游戏中。

游戏发布后，一些爱好者将Doom的技术规格发布出来帮助人们制作游戏里的数据物件（是指玩家通过替换数据文件创造出新的游戏场景、角色和装备等等——译者注），id公司对此表现得很友善，他们有时会直接和黑客合作，回答黑客的具体问题，发布内部的规格文档，鼓励人们制作新的Doom数据并发布到互联网上。

技术趋势和市场趋势使得源码开放的收益提高，Doom开放技术规格和鼓励第三方扩展（add-on）的做法，都提高了游戏的感知价值 [5]，并打开了可供利用的二级市场。开源和闭源所对应的收益曲线会在某一时间点上交叉，此时，将源码开放并从二级市场上（如游戏场景选集等产品）赚钱会成为id公司的经济理性选择，确实如此，1997年末，id公司将完整的Doom源代码公开发布。

知道何时放手

Doom是一个有趣的案例，它不是一个操作系统也不是一个通信/网络软件，它和常见的开源成功案例明显不同。事实上，Doom的整个生命周期（包括从闭源转向开源的过程）可能会成为当今应用软件代码生态的典型——在这个生态里，信息交互和分布式计算带来了严重的健壮性/可靠性/可扩展性问题（只能靠同行评审来对付），并常常跨越技术环境界限和竞争者界限（其中隐含着信任和对等问题）。

Doom从单人游戏演化到多人对战游戏后，网络效应开始大显身手。其他领域也是如此，随着商务网络（当然它位于整个WWW架构之中）正空前紧密地把供应商和客户联系在一起，这一点甚至在最重量级的商业应用（如ERP系统）中也能看到。相应地，开放源码的收益也会因此稳步提升。

如果这个趋势继续下去的话，21世纪软件技术和产品管理的核心挑战将是明白何时放手——即何时允许把封闭源码转为开放源码体系，利用同行评审的好处，在服务上和二级市场上获取更高回报。

很明显，出于收入上的考虑，在任何一条曲线上都不应走得离交叉点太远。但是，等待太长时间会面临严重的机会风险——市场上的对手可能会抢先开放源码。

这个问题之所以严重，是因为对任意给定种类的软件产品，开源合作能够吸引的用户群和专家群都是有限的，而社区往往有黏性，如果两个在功能上大致等同的产品先后开源，先开源的往往会吸引最多的用户和最有激情的合作开发者，后开源的只能吃剩饭。社区之所以有黏性，是因为用户对软件已经熟悉，而开发者已经在代码上投入了太多时间。

4.11　用开放源码做战略武器

开源不仅可用来拓展市场，有时还可作为一种有效对抗公司间竞争的战略手段。从这个角度重新检视一下前面提到的一些商业战术，会有很多新发现，开源此时的直接目的不是为了收入增长，而是为了闯入市场和重塑市场。

用成本分摊做竞争武器

前面谈到的Apache，是一个在开源项目中通过成本分摊实现质量更好和成本更低的基础架构开发的例子。对于想要和微软及其IIS服务器竞争的软件及系统供应商，Apache项目都可以是一个竞争武器。因为任何单个的Web Server供应商，想抗争微软所拥有的巨量“战争基金”（war chest）以及桌面市场上的垄断能力，都是艰难甚至不可能的。而Apache项目使得每个参与者能够以低得多的成本，提供一个不但在技术上优于IIS而且能让客户放心使用的拥有大部分市场份额的Web Server产品。而且，Apache还改善了增值电子商务产品（如IBM的WebSphere）的市场地位和生产成本。

概括来说，基础架构的开放和共享，使每个参与者都得到了竞争上的好处，一是参与者能以较低成本生产出可扩展的产品和服务，二是参与者的市场定位可以让客户放心，他们很少会面临这样的尴尬境遇：由于供应商更改了战略或战术，导致产品被抛弃而无人照管。

重置竞争

DEC公司在20世纪80年代资助X窗口开源开发的明确目标是“重置竞争”。当时，市场上有若干个相互竞争的UNIX图形环境，其中包括Sun公司著名的NeWS系统。DEC的策略家认为，如果Sun建立起一个私有的图形标准（这很有可能），Sun就能够对正在蓬勃发展的UNIX工作站市场施加影响。通过资助X并将它贡献出来，DEC可以联合若干小供应商将X推成事实标准，从而抵消Sun和其他竞争者在图形专业知识方面的优势。这将使工作站市场的竞争转向硬件层面，而硬件可一直是DEC的强项。

这一点具有普适性。开源能够吸引精明的客户以及那些还没有足够大到可以独自投资竞争性开发的潜在盟友。适时展开的开源项目比仅仅只是在产品上超越闭源对手更有效一些；开源项目可以有效防止闭源产品制约市场，能重置竞争并让新进公司从劣势领域转至优势领域里。

扩大水池

为了给Linux世界一个标准的二进制包安装程序，红帽公司（Red Hat Software）资助了RPM软件包的开发。通过这种方式，他们确信这种标准化的安装程序会给潜在客户增强信心，并因此带来更好的未来收入，尽管会有软件开发成本，并且会因为竞争对手也使用它而损失一些潜在收益。

有时候，要想成为一只更大的青蛙，最佳办法就是让水池更快变大，这就是技术公司参与公开标准（完全可以将开源软件看成是可执行标准）的经济原因。除了成为一个卓越的市场建立者外，这种策略可以

被小公司当做直接的竞争武器，去对抗一个标准联盟之外的拥有市场控制力的大型公司。在Red Hat的案例中，明显且公认的强有力对手是微软，在大多数Linux发行版中推广RPM，势必会很有效地抵消微软的优势：后者在Windows机器上拥有便利的系统管理工具。

防止锁喉

在前面解释“占领市场”商业模式时，我介绍了Netscape将Mozilla浏览器开源是阻止微软对HTML和HTTP进行事实上控制的一种成功手段。

通常，防范竞争对手将某项特定技术牢牢“锁喉”要比你自己控制该技术更重要。通过开源，你能组成规模极为强大的防范联盟。

4.12 开源和商业战略风险

开源似乎注定要成为一种普遍的做法，究其原因，更多是源自于客户需要和市场压力，而非供应端的效率。我已经从供应商角度讨论了客户对可靠性以及对基础设施市场不存在垄断者的要求会带来什么样的效果，也说明了这些因素如何在网络演变史上发挥作用。当市场中存在开源竞争者，客户行为会有更多话题可谈。

设想你现在是一个财富500强企业的CTO，你正在考虑建设或升级你们公司的IT基础设施。也许你需要选择一个网络操作系统并将其部署在整个公司；也许你关注24/7的Web服务和电子商务；也许你的业务依赖于高吞吐量和高可靠性的交易数据库。

如果你走传统的闭源路子，就是让你的公司听由供应商的控制和摆布——就其定义来看，你只能在它那里寻求支持、bug修复和改进。如果这个供应商不配合，你会没辙，因为你已经被初期的投资和培训成本套牢了，而供应商也知道这点。这种情况下，你觉得软件会被继续改进以满足你的需求和你的商业计划，还是去满足供应商的需求和他的商业计划？

事实残酷之处在于：你的关键业务流程执行在不透明的二进制比特上，你看不见里面（更别说修改了），你已经失去了对业务的控制。你的供应商需要你，但你更需要他——你会不停的付款、付款、再付款，为的就是获取权利的平衡。当供应商收紧缰绳的时候（他们早已熟谙这

种和受害者之间的游戏），你会为更高的价格付款，为失去的机会付款，为这种日益恶化的套牢效应付款。

对比一下选择开源的情况。如果这样做，你手里有源代码，而且没人能拿走它，没有哪个垄断供应商能对你的业务锁喉，而且会有多个服务商为你的业务竞标——你不仅可以让他们互相竞争，如果觉得外包不划算，你还可以组建隶属于你的支持团队，市场是为你服务的。

这个逻辑是令人信服的，依赖闭源会带来无法接受的战略商业风险。以至于我相信，不久之后，在有可行开源方案的前提下，仍然采购单一来源闭源产品的行为将会被认为是一种不尽职，并会成为股东起诉的正当理由。

4.13　开源的商业生态

开源社区以一种倾向于增强其生产效能的方式自我组织。在Linux世界里，若干个相互竞争的Linux发行商形成一个与开发者相隔离的层级，这是一个很有经济意义的现象。

开发者写代码，并使其可通过互联网获取。每个发行商从这些代码中选择若干，将它们集成、打包并使用自己的商标，然后销售给用户。用户选择发行版，并可能从开发者的网站直接下载代码，以补充发行版的不足。

这种层级隔离的做法，创建了一种非常灵活的内部市场[6]改善机制。开发者在软件质量上互相竞争，为的是获取发行商和用户的注意。发行商在选择策略上以及给软件带来的附加价值上竞争，为的是从用户那里赚钱。

内部市场结构的首要效应是：这个网络中没有哪个节点是必不可少的。开发者可以退出，即使没有其他开发者接手他的那部分代码，出于对注意力的竞争，往往也会很快产生该软件在功能上的替代品。发行商可能会失败，但它并不能损坏或危害公共的开源代码库。这种生态作为一个整体，对市场需求有着更快的反应速度，能更好地抵抗冲击和自我更新，这是任何单体结构的闭源运行体系都不能企及的。

另一个重要的效应是通过专业化减少开销并提升效率。传统闭源项目给开发者带来的压力，会日复一日地损害软件质量并将项目变成“焦

油坑”（tar-pit）[7]，开源开发者则无需承受这种压力——没有来自市场部的无意义且分散注意力的功能列表；没有管理层要求他们使用不合适和过时的语言或开发环境；不会因为产品差异化或知识产权保护的需要而以一种新的、不兼容的方式重新发明“轮子”；（最重要的是）没有“最后期限”；不需要在产品还没有做好时就匆匆忙忙推出1.0版本。De Marco和Lister在“Peopleware:Productive Projects and Teams”（见书后“大教堂与集市”的注释5）中指出，“好了告诉我”管理模式不仅有利于提升质量，实际上也有利于工作成果的最快交付。

另一方面，发行商专注于他们最擅长的事情。不需要为了跟得上竞争而为大规模和持续的软件开发去筹集资金，他们可以集中精力于系统集成、打包、质量保证和服务。

作为开源方法不可或缺的一部分，用户不断地反馈和监督，发布者和开发者则因此保持相应的诚信。

4.14　应对成功

“公地悲剧”可能不适用于如今的开源开发项目，但并不意味没有理由去怀疑当前开源社区的发展势头是否可以持续。当利益砝码越来越重时，主要参与者是否会叛离合作？

这一问题可以从不同层面提出。我们的“反公地”模型构建在这样的论点上：个体对开源做出的贡献很难被货币化。但这一论点对那些已经从开源获取收入流的公司（比如说Linux发行商）缺乏效力。他们的贡献每天都在被转化为货币，他们作为合作者的角色稳定吗？

探究这个问题可以让我们对一些很有意思的问题进行思考，比如在现实世界中的开源软件经济学，以及经典服务行业能给软件行业的未来带来什么启示。

在现实层面上讲，这个问题已经反映在开源社区，并常以两种不同的方式被提出。第一种是“Linux会分裂吗”，第二种则相反，“Linux会成为一个占据统治地位的准垄断操作系统吗？

谈到Linux是否会分裂，很多人会拿上世纪80年代专有UNIX供应商的行为做类比，尽管就开放标准问题进行了无数次会谈，尽管成立了众多的联盟、团体和协定，专有UNIX还是四分五裂了。事实证明，供应商更有兴趣通过增加和修改操作系统功能将其差异化，而不是考虑如何通过维护兼容性把UNIX市场整体做大（从而降低独立软件开发者的进入壁垒，并减少客户的拥有成本）。

这不太可能会发生在Linux身上，原因很简单，所有发行商都不得不在同一个公用开源代码库上操作。他们中的任何一个都不可能在事实上保持差异化，因为许可证（Linux代码的有效成长得益于此）要求他们将代码与其他各方共享。任何发行商一旦开发出一个新功能，所有竞争者都可以免费克隆它。

由于各方都了解这一点，人们连想都不会去想那些曾经将专有UNIX分裂的手段。取而代之的是，Linux发行商必须要以一种实际上有益于客户和整个市场的方式去竞争。也就是说，他们必须在服务、支持以及设计上竞争，看谁在实际上能提供更容易安装和更容易使用的界面。

公用源代码库还排除了垄断的可能性。当Linux社区的人担心垄断时，他们常常提到“Red Hat”，它是最大和最成功的发行商（在美国占据大约90%左右的市场份额）。但值得注意的是，1999年5月，期盼已久的Red Hat 6.0宣布发布后，几天之内——在Red Hat的CD-ROM还未真正发货之前——从Red Hat的公开FTP站点下载并构建的该版本CD-ROM，已经被一家图书出版商和其他一些CD-ROM发行商开始以比Red Hat预期售价更低的价格开始宣传销售了。

Red Hat对此不动声色，因为其创立者非常明白，他们没有也不能拥有产品中的任何一个比特，Linux社区里的社会准则不允许这样。John Gilmore后来有一个著名的论断，即互联网把网络审查看做是对自己的损害并且会绕开它。对Linux黑客社区而言，很恰当的对应是，Linux把控制企图看做是损害并绕开它。Red Hat如果抵制对他们最新产品的抢

先克隆，将会严重危及他们未来从开发者社区吸引合作者的能力。

也许目前更重要的是，建立有法律约束力的能够表达社区准则的软件许可证，防范Red Hat对产品所基于源代码的垄断。Red Hat唯一能出售的是品牌、服务和支持，人们则自愿为此付费，这样就不会使掠夺性垄断有太大的可能。

4.15 研发开放和资助创新

向开源世界注入资金的另一个考虑是改变开源的现状。开发社区明星越来越多地发现他们可以从自己想做的事情中获取报酬，而不是像以前必须要通过正式工作收入才能维持自己对开源的爱好。Red Hat、O'Reilly&Associates和VA LinuxSystem这类公司正在建立类似半独立的研究机构，他们通过合同雇佣开源人才并保持其稳定性。

若想让这种做法产生经济意义，公司必须采用快速扩大市场的办法来获取预期收益，以支付维持实验室运转的人力成本。O'Reilly之所以能够负担起Perl和Apache领导人的薪水，是因为预期他们的工作能够让公司出售更多Perl和Apache相关的书籍，并吸引更多人参加其会议；VA Linux System之所以资助实验室是因为改进Linux会提高他们所卖工作站及服务器的使用价值；Red Hat之所以创建Red Hat高级开发实验室是因为可以借此提升他们Linux产品的价值并吸引更多的客户。

传统软件产业的战略家们是无法理解这种行为的（不顾其市场增长效应），因为他们成长在将（通过专利和商业秘密保护起来的）知识产权看成企业王冠上宝石的文化之中。为什么资助一项研究，却让每个竞争对手都可以无偿享用其结果呢？

对此有两个主流解释。一是只要这些公司继续在其市场生态中保持领先地位，他们就可以从开放研发中获得最大份额的回报，投入研发以期获取未来利润并不是什么新奇想法，有意思的是其背后的打算：由于

预期未来收益足够多，这些公司为了同行评审效果而对“搭便车”者大度容忍。

“预期未来收益”这种直白的分析，对于把眼睛一直盯在投资回报率上的顽固资本家来说是必要的，但它并不是最有趣的解释。那些“雇佣明星”（star-hiring）的公司高管们也给出了一个较为模糊的解释，如果问他们为什么这么做，他们会说，这只是在做社区认为正确的事情，而他们本人正来自社区。鄙人和以上所提三个公司的高管都非常熟悉，我可以作证他们这番言论绝不是虚妄之言。事实上，1998年末，我本人被正式聘为VA Linux Systems董事会一员并因此可以建议他们“做正确的事”，我发现他们对此并非没有兴趣。

经济学家可能会问，这样做会有什么收益？如果我们认可“做正确的事”不是空洞的摆姿态，接下来就该问一问，“做正确的事”会给公司带来什么好处？答案本身既不令人惊讶也不难以验证，其他产业里也有这种看上去大公无私的行为，这些公司相信他们换来的是名声。

争取名声并将其视为能从未来市场获取回报的无形资产，这种想法并不新鲜，真正有趣之处在于，这些公司的行为表明他们肩负起了道义，而这将会给公司带来极高的估值。即使是在IPO准备阶段资本非常匮乏之时，他们仍然明确表示愿意以高价请到高人，从事一些并不能直接产生收入的项目。目前看来，市场已经非常丰厚地回报了这种行为。

这些公司的高管们非常清楚名声有着特别的价值。公司倚重来自客户群中的志愿者们，因为他们不仅开发产品，也是公司非正式的市场营

销力量。这样，公司和客户群的关系会更加亲密，而且常常是建立在个体之间的相互信任关系上（不管在公司内部还是外部）。公司不只是在利用黑客社区，他们和黑客社区在根本上是一体的。

这些现象又一次强化了我们之前从另一条推理路线得出的结论。RedHat/VA/O'Reilly这些公司和客户/开发者之间的亲密关系在制造业并不常见，而更像是高度专业化和知识密集型服务产业的特有模式，并且将这种模式推向了引人入胜的极致。除科技产业外，该模式还可在法律事务所、医疗执业和大学这类机构中看到。

我们应该看到，事实上，开源公司聘请明星黑客和大学聘请明星教授的原因大致相同，他们的做法在机制和效果上类似于工业革命前贵族对高雅艺术的资助——部分当事人完全了解这种相似性。

4.16　走向彼岸

资助开源开发（并从中获利！）的市场机制仍然在快速发展之中。本文所列举的商业模式并不是最终定式，投资人还在不断思考如何应对软件产业变革带来的后果，软件产业变革后将明确定位于服务而不是封闭的知识产权，这一天终将到来。

这种概念上的革命会让那些靠销售价值盈利（占产业的5%）的投资者们有所损失，从历史上看，服务业的利润不如制造业那么丰厚（但任何一个医生或律师都会告诉你，实际从业者所获的回报往往会更高），然而，当软件消费者可以从开源产品中获取惊人的节约和效能时，投资者从中获取的好处会远超其损失（类似的例子是将传统语音电话网络置换为无处不在的互联网）。

这种节约和效能的承诺正在创造一个市场机会，而企业家和风投家也已开始利用这个机会。在本文第一版草稿酝酿之时，硅谷最著名的一家风险投资机构下了头注，投资给第一家提供24/7专业化Linux技术支持的创业公司（Linuxcare）。1999年8月，Red Hat的IPO大获成功（尽管是在互联网和科技股暴跌的背景下），人们普遍预期，1999年底之前，会有若干家Linux及开源相关的IPO将会展开——他们也会非常成功。（2000年更新：确实如此！）

另一个很有趣的发展方向是：在开源开发项目中尝试系统性地建立任务市场。

SourceXchange（http://www.sourcexchange.com/process.html）和CoSource（http://www.cosource.com/）都试图通过反向拍卖来资助开源开发，虽然在方式上略有不同。

整体趋势是很明显的，我们前面提到IDC预测Linux在2003年之前的发展速度会比其他操作系统的总和都要快。Apache已经占到市场份额的61%而且还在稳步增长。互联网的使用是爆炸式的，诸如Internet Operating System Counter（http://leb.net/hzo/ioscount/）这类调查报告显示Linux和其他开源操作系统已经是互联网主机所采用的主流系统，而且其市场占有率（比起闭源系统）还在不断增大。充分利用开源互联网基础架构的需求不仅对软件设计，也对商业实践以及每个企业使用和购买软件的模式产生越来越重要的影响。这个趋势，似乎还正在加速。

4.17　结论：变革之后

在完成开源变革之后，软件世界会变成什么样？

一些程序员担心开源运动会使他们失业或贬值，最常见的恶梦是我所称的“开源审判日”（Open-Source Doomsday）：慢慢地，到处都是免费的源代码，软件的市场价值趋向于零，单靠使用价值已经无法吸引足够的消费者支持软件开发，商业软件产业土崩瓦解，程序员挨饿或者改行，最后，到开源文化（依赖于这些程序员的业余时间）自身也崩溃的时候，审判日就来了，再没人能胜任编程工作，全都完了，哎，真是恶梦！

这并不会发生，对此我们有充分的理由。首先绝大多数开发者的薪水并不取决于软件销售价值，但最好的也是最值得强调的理由是：你什么时候见过一个软件开发团队的活不够干？在这个快速变化的世界里，经济社会日益复杂并以信息为中心，懂计算机的人可要有个好身体，因为总有很多活等着他们做——无论他们花了多长时间，传授了多少诀窍。

为分析软件市场自身，有必要将软件服务按技术标准化程度进行分类，而这和软件服务的市场化（commoditize）程度存在密切关系。

这种分类与人们通常所说的“应用”（完全没有市场化、已开放的技术标准太弱或不存在）、“基础架构”（市场化服务、强标准）和“中间件”（部分市场化、有效但不完全的技术标准）有着很好的对应。在今

天，典型的例子是字处理软件（应用）、TCP/IP协议栈（基础架构）和数据库引擎（中间件）。

我们早先所做的收益回报分析表明：基础架构、应用和中间件将会以不同的方式变革，并展现出不同的开、闭源并存及平衡现象。在特定软件领域，开源能否流行，将取决于软件是否有实质性的网络效应、软件失效的代价如何以及软件作为资本货物的业务关键性程度。

如果将这个启发式分析方法运用到软件市场的各个部分（而不是单个产品），我们可以做出如下的大胆预言：

基础架构（互联网、Web、操作系统、跨越竞争者界限的低层通信软件）将会几乎全部开源，并由用户联盟和盈利性发布/服务机构（如Redhat所扮演的角色）共同维护。

应用，则非常倾向于继续封闭。当一个未公开算法或技术的使用价值足够高（且软件不稳定带来的相关成本足够低、供应商垄断带来的相关风险足可容忍）时，用户会继续为此类闭源软件付费。这种情况最有可能发生在自成一体的垂直市场应用中（其网络效应也较弱）。前面提到的锯木软件就是一例，1999年最热门和最有前景的生物识别软件则是另一例。

中间件（像数据库、开发工具或可定制的应用协议栈顶端）将处于开闭源混杂的状态，这类软件走向闭源还是开源，似乎更取决于软件失效的代价，代价越高，其走向开放的市场压力就越大。

更全面地看，我们需要注意到“应用”和“中间件”都不是稳定的类

别。前面我们已经看到，单个软件技术看上去会自然而然历经从理性封闭到理性开放的生命周期，这个逻辑对软件类别一样适用。

应用会倾向于落入中间件之列，因为技术不断标准化，而服务也逐步市场化（比如，在SQL与引擎前端解耦后，数控库就成了中间件）。当中间件市场化后，他们就倾向于落入开源基础架构之列——正如我们现在所经历的操作系统变革。

可以预期，由于开源的存在，任何软件技术的最终命运不是灭亡就是成为开源基础架构的一部分。这对那些想靠闭源软件永远收取租金的企业家来说可不是好消息，但软件产业从整体上仍会保持企业性质，上层（应用）软件将不断走向开放，封闭知识产权垄断的时间越来越有限，因为他们的产品终将落入基础架构之中。

最后，这种态势显然对软件消费者（正是他们驱动着这一进程）非常有利。越来越多的高质量软件被创造和使用，而不是被中止和闭藏。用Ceridwen女神的魔法锅来比喻实在是太弱了——因为食物要么被吃掉要么烂掉，而软件源码将会永存。自由市场，在其最广阔的自由意义上讲，只要不是强制行为，不管是通过市场交易还是礼物馈赠，都会给每个人带来源源不断的软件财富。

4.18　后记：为什么驱动程序闭源会让制造商坐失良机

计算机外围硬件（如以太网卡、磁盘控制器和显卡等等）制造商在传统上是抵触开放的。但现在情况有所改变，像Adaptec和Cyclades公司已经开始逐步公开自己的产品规格和板卡驱动的源代码。尽管如此，抵制开源的心理仍然很普遍。下面，我会尝试从经济层面分析这种心理背后的种种错误观念。

如果你是硬件供应商，你可能害怕开源会泄露产品如何工作的重要细节，会使竞争者趁机复制并获得不公平的竞争利益。在产品周期长达3年到5年的那些年代里，这个观点还算说得过去，但现如今，竞争者花在复制和理解上的时间，将会占据产品周期的很大一部分，他们本该把时间花在创新和考虑如何让产品差异化的。

这个观点并不新鲜，前克格勃首脑Oleg Kalugin说得很好（http://cnn.com/SPECIALS/cold.war/experience/spies/interviews/kalugin/）

“比如说，我们计划窃取IBM这类公司或者其他电子领域的先进技术，由于西方在这方面远超我们，我们需要花费数年让这些情报成果得到实现。而那时，大概五年或七年，西方又往前走了，我们只能跟着一偷再偷，而且会落得越来越远。”

Rudyard Kipling在几乎一个世纪前更好地阐明了这点，在诗作“The

Mary

Gloster”（http://www.everypoet.com/archive/poetry/Rudyard_Kipling/kipling

中，他写道：

他们问我如何做到的，

我把圣经给他们看，

“让你的光继续闪耀，

照亮在跟随者的前方！”

他们会设法复制一切，

却无法复制我的思想，

我让他们辛苦偷窃，

却永远落后我一年半载。

在一切都变得更快的互联网时代，这种效应更明显。如果你确实走在前面，他们的剽窃会正中你的下怀！

不管怎么说，如今代码中的技术不可能隐藏太久。硬件驱动程序不像操作系统或者应用，它们很小，很容易被反编译和克隆。甚至十多岁的新手程序员也能干这个——常常真的如此。

不夸张地说，现在有数以千计的Linux和FreeBSD程序员正在摩拳擦掌，他们有能力也有动力给新的板子写驱动。对于那些接口相对简单或者已经标准化的设备（如磁盘控制器和网卡），这些充满激情的黑客们，写起驱动可一点儿也不比厂家慢，即便在没有文档也不去反编译现有驱动的情况下。

即便像显卡和声卡这种棘手的设备，你也没有什么办法阻止一个装备了反编译器的聪明程序员去做点什么。反编译的成本较低，并且有办法绕过法律壁垒，由于Linux是全球性的工作，为它而做的反向工程，通常都被判定为合法的。

如果要找支持以上观点的有力证据，可以检查一下Linux内核支持的设备列表，并注意一下新设备的增加速率，要知道，这往往是没有供应商支持的。

开放驱动程序源码的另一个好处是你可以集中精力放在创新上。想象一下你不再需要把员工时间和薪水花在重写、测试和发布新的二进制文件上(每当新内核面世时)，你有这样的技术能力，当然要做更有用的事。

还有一个重要原因：没人愿意为了一个bug的修复而等上半年，你会仅仅因为有个开源竞争对手而惨遭用户抛弃。

当然，还有我们前面提到的“防患于未然”效应。客户之所以希望开源，是因为他们知道这会延长硬件的使用时间，而不是考虑让你在支持成本上更划算。

最重要的理由是：你是靠销售硬件赚钱的，没有客户想要你软件中的秘密。事实上，如果你的驱动很难找，如果它不得不频繁更新，如果它运行得很差（这一点最糟糕），都会给硬件带来恶劣影响，你的硬件会卖不动。而开放源码能解决这些问题并提升你的收入。

这说明什么？在短期内看，将驱动程序保密的做法似乎还行；但从

长远看，这可能是个糟糕的策略（尤其当竞争对手已经开放源码时）。如果你不得不保密源码，可以把代码烧到板子上的ROM中，然后发布ROM接口。你要尽可能地开放，要建立市场并向潜在客户展示你的自信——你在思想上和创意上超越竞争者的自信。

如果你坚持闭源，你可能要面对最坏的结果——你的机密还是会被曝露，你不会得到免费的开发帮助，你也别指望你的对手会傻到花时间克隆你的东西。最重要的是，你错过了及早获取认可的阳光大道。一个巨大而又有影响力的市场（那些有效运转整个互联网和多数商业数据中心的服务器管理员们）会因为你的无可救药和顽固抵抗而把你的公司打入冷宫。你不懂为什么要开放，没关系，他们会从懂的人那里买板子。

[1] 原文是ephemeralization，是R.Buckminster Fuller创造的术语，他认为技术进步的趋势是更小、更轻和更有效，能够让人们“费力越来越少，收获越来越多，以至于最终可以毫不费力地获得所有东西”。这种观点认为，随着技术进步，即便地球资源有限，也能提供给全世界日益增长人口以高质量的生活。“以少成多”这种趋势使得人们能以较低的成本获得无限的产出。Fuller曾举例说明，通讯卫星只有几百斤重，却可以在功能上完全替代重达75万吨的海底电缆。——译者注

[2] 竞次（race to the bottom）是一个社会经济学概念，常常用在反全球化和支持公平贸易的语境中。通常是指在全球化自由贸易背景下，国家之间或地域之间在某种贸易或产品上的竞争变得日益激烈，为赢得竞争中的价格优势，政府在商业法规、劳工标准、环境保护、工商税务上

放松管制，导致劳动保障变差、工人工资变低，自然环境损害等负面效果。这种逼近底线的竞争，比的不是谁更优秀，而是比谁更次、更糟糕、更能够苛刻本国（地域）的劳动阶层，更能够容忍对本国（地域）环境的破坏。——译者注

[3] 在经济学和政治学分析中，搭便车者（Free r ider）是指那些在资源使用或项目参与活动中，获取了多于应得利益或是不承担自己应付成本的人，尤其是那些事先宣称自己并无需要，但在别人付出代价取得后，却不劳而获享受成果的人。搭便车行为会妨碍市场的自动调节过程，导致市场失灵和效率变低。——译者注

[4] 网络效应（network effect，也称为网络外部性）是指某商品或服务的已有用户数对用户从该商品或服务中获取价值所产生的效用。梅特卡夫定律（Metcalfe's law）指出，如果某个商品或服务拥有网络效应，其总价值大约和已经拥有该商品或服务的客户数量的平方成正比。——译者注

[5] 感知价值（perceived value）是客户所感知的某个产品对自己的价值，取决于产品对客户需求的满足程度，是客户对产品价值的主观认知。感知价值不同于传统意义上的客户价值概念，后者是指企业认为自己的产品可以为顾客提供的价值，而前者是指客户对企业所提供产品的价值判断。——译者注

[6] 内部市场（Internal market）是指在一个（或多个）组织内，各成员单位通过经济活动而形成的市场。这些成员单位是解耦的，每个单位

向组织内其他单位交易自己的服务。这些服务可能只面向内部市场，也可能面向公开市场。——译者注

[7] 焦油坑（tar-pit），又译“沥青坑”或“沥青湖”，是指地表中天然沥青的积集场所，尤指使动物不慎陷入其中并保存了它们的骨骼的天然沥青聚集点。从地层中冒出来的石油干涸后，只留下半固态的焦油沥青，烈日照射下，焦油变软，无论什么东西接触到它，就永远陷入其中。《人月神话》中谈到：“史前史中，没有别的场景比巨兽在焦油坑中垂死挣扎的场面更令人震撼。上帝见证着恐龙、猛犸象、剑齿虎在焦油坑中挣扎，它们挣扎得越是猛烈，焦油就纠缠得越紧，任何猛兽都没有足够强壮的体力或技巧来挣脱束缚，它们最后都沉到了坑底。”——译者注

5.黑客的反击

1998年，开源软件经历爆发式增长后开始进入主流，这可看作是黑客对20多年来边缘化地位的反击。我发现自己有意无意间已经被人们视为这场反击的首要煽动者和宣传者。本文我描述了随后这喧哗的一年，并着重介绍我们让开源产品打入财富500强的媒体策略和话语方式。最后，我对这股潮流的趋势做了展望。

5.1 黑客的反击

我在1996年写下“黑客圈简史”第一版并通过Web发布，那时，我着迷于黑客文化已经很久了，最早要追溯到1990年我编辑“新黑客词典”第一版时。到1993年末，很多人（包括我自己）已经开始把我当做黑客文化的部落史学者和居民人种学者了，我很满意这个角色。

那时，我完全没有想到，我所做的这个业余的人类学研究对软件变革有着重要的催化作用，当变革发生时，我想没有人会比我对此更感到惊讶了。变革的影响至今仍然在黑客文化界、技术界及商界回响不息。

我将从个人视角出发，概括那些直接导致1998年1月开源革命“惊世一枪” [1]的事件，我将反思我们所走过的非凡历程，并小心翼翼地给出一些对未来的预测。

5.2　超越Brooks定律

1993年底，通过最早的CD-ROM发行版Yggdrasil，我第一次接触到Linux。那时我已经被卷入黑客文化15年了，早在20世纪70年代末，我就开始接触早期的ARPAnet，我甚至还简单地见识过ITS机器。在我开始写自由软件并将其发布到Usenet上时，自由软件基金会（FSF）还未成立，1984年FSF成立后，我是它的首批贡献者之一。最近我刚发布了“新黑客词典”的第二版，我想我懂黑客文化（包括它的局限性），而且很懂。

正如我在别处所写的那样，Linux让我大吃一惊。虽然那时我在黑客文化中已经活跃多年，我仍然抱有一个未经检验的假定，认为黑客尽管可能很有天赋，但若要制作一个可用的多任务操作系统，他们没有能力调动所需的资源或技术。HURD开发者历经十年来的努力，最终证明了这一点。

但就在他们失败的地方，Linus Torvalds和他的社区成功了，他们不仅实现了稳定运行UNIX接口的最低需求，而且完全超越了那个标准，他们用自己的活力和才华，提供了数百兆的程序、文档以及其他资源。他们贡献了整套的互联网工具、桌面出版软件、图形支持、编辑器、游戏……乃至所有你能想到的东西。

看到这些精彩绝伦并能组成可工作系统的代码盛宴，其震撼体验远远超过仅在理智上知道这些比特的存在。就好比多年来我一直在整理成

堆的汽车零配件，突然间，面前出现一辆由同样元件组装起来的闪闪发亮的红色法拉利，门开着，钥匙在锁上摇摆，引擎温柔地轰鸣，承诺着它将给予的能量……

20多年来我所观察到的黑客传统，似乎瞬间有了一种充满能量的新活法。在某种意义上，我已经成为这个社区的一部分，因为我的若干个人作品已经作为自由软件加入其中，但我想更深入一些，因为我每次感受到惊喜时，都又一次加深了我的困惑：它怎么会这么棒。

在软件工程的全部经验中，Brooks定律占据着统治地位，它来自于Fred Brooks的经典著作《人月神话》。

Brooks推测，如果开发成员数目为N，工作量会呈N倍增长，但复杂性和bug率会以N^2增长。N^2体现着各开发者代码之间的通信路径（以及可能的代码接口）。

对于一个有数以千计贡献者的项目，Brooks定律预言它必然会成为怪异不堪、难以驾驭的一团乱麻，但不知怎地，Linux社区打破了N^2效应，他们做出了质量高得惊人的OS。我决心要弄明白这是怎么回事。

我花了三年时间亲自参与和近距离观察，最终得出一个理论，并用另外一年去实际检验它。然后我坐下来，写下了“大教堂与集市”，来说明我所看到的一切。

5.3　模因[2]和神话创造

我所看到的是：这个社区发展出有史以来最有效的软件开发方法却浑然不觉！也就是说，这个有效实践在不断发展中已经逐渐形成了一套习惯，并通过模仿和示例来传播，却没有理论或语言来解释它为什么有效。

事后看来，理论和语言的缺乏在两个方面对我们不利：一是我们不能系统性地思考如何改进我们的方法，二是我们难以向别人解释和推销我们的方法。

当时我思考的只是前者，我写那篇文章的唯一目的，就是给黑客文化一个合适的可供内部使用的语言，以便向自己解释。所以我把我看到的写下来，采用叙事框架，辅以恰当和生动的比喻，并描述这些习惯背后的逻辑。

“大教堂与集市”里没有根本性的发明创造。我没有发明其中的任何方法，新鲜的不是事实，而是比喻和描述——用简单有力的故事鼓励读者以新的方式看待事实。我只是尝试把些许模因工程的方法，用在黑客文化那充满生产力的神话上。

1997年5月，在巴伐利亚召开的Linux Kongress大会上，我首次发表了整篇文章，并受到了高度关注和雷鸣般的掌声（虽然听众中很少有人以英语为母语），这证明我可能真的对了。碰巧的是，周四晚宴我正好坐在出版人Tim O'Reilly的旁边，这引发了一系列更有意义的后果。

我对O'Reilly的行事风格仰慕已久，寻求与他的会面机会已经好几年了。那天我们进行了广泛的交谈（更多是谈论我们的共同兴趣——经典科幻小说），并直接导致我被邀请在年底Tim举办的Perl大会上发表“大教堂与集市”。

这篇文章又一次受到热烈欢迎——听众们起立鼓掌并喝彩。事实上，自巴伐利亚之后，我从邮件得知，有关“大教堂与集市”的议论已经如星火燎原般在互联网上广为传播。一些听众已经读过文章，演讲内容对他们来说已经不新鲜，他们更多是为了庆祝这新的言论和随之而来的意识觉醒。听众们的起立鼓掌，与其说是献给我，不如说是献给黑客文化本身——这是理所当然的。

我还不知道我在模因工程上的这次实验，将会点燃更巨大的火焰。有些听众来自网景通信公司，他们感觉我演讲的内容非常有创意，而网景当时正陷在麻烦之中。

网景，作为互联网技术公司的先驱和华尔街炙手可热的明星，正被微软视作必除之而后快的眼中钉。微软当然害怕网景浏览器所代表的开放Web标准会损害Redmond巨人（Redmond为微软总部所在地——译者注）在PC桌面上丰厚的垄断利润。微软强大的资金和其充满争议的战术（这导致随后的反垄断诉讼）都已部署好，目标就是搞垮网景浏览器。

对网景来说，他们并不太关心浏览器相关的收入（这永远是其收入的很小一部分），他们更关心如何给更具价值的服务器业务营造一个安

全的空间。如果微软的IE浏览器主宰了市场，微软将会扭曲开放标准的Web协议，将其带入专有化路径，进而使得只有微软服务器才能提供Web服务。

关于如何对抗微软，网景内部有着激烈的争论。其中一个提议是把网景浏览器开源——但这个提议缺乏有力的理论支持，人们很难相信这样做就能阻止IE占据主导优势。

那时我还不知道，“大教堂与集市”这篇文章成了赢得争论的主要因素。1997年冬天，在我为下一篇文章准备材料时，网景公司开始着手打破专有软件的游戏规则，并给黑客部落提供了一个前所未有的机会。

5.4　山景城之行

1998年1月22日，网景宣布将会在互联网上发布网景客户端产品线的源代码。很快，第二天我就得知消息，说CEO Jim Barksdate当时对媒体记者讲，正是我的工作，给予了这项决策以“主要灵感”。

这就是计算机行业刊物评论员后来所称的“惊世一枪”——不论我是否愿意，Barksdale把我当成他的Thomas Paine [3]。对黑客文化而言，在历史上第一次，一个受华尔街宠爱的财富500强公司，把它的前程赌在了相信“我们的路是对的”，特别是赌在了相信我对“我们的路”的分析。

这件事让我震惊并让我开始冷静思考，“大教堂与集市”一文改变了黑客文化对自身的印象，对此我并不奇怪，毕竟这正是我所寻求的结果。让我惊讶的是，它成功的消息居然来自黑客文化的外部。在得知这条消息的头几个小时内，我开始认真思考Linux和黑客社区的现状，思考网景的问题，以及我个人下一步应该做些什么。

不难得出结论，帮助网景这次的孤注一掷，是黑客文化当前优先级非常高的事件，对我个人亦然。如果网景的押注失败了，很可能所有恶名都会向我们袭来，我们将会在下一个十年里不被信任，那我们承受的代价就太大了。

我沉浸在黑客文化中已经20多年并且历经了它的各个时期，20年来我看到闪光的想法、充满希望的开端和高超的技术一次又一次被圆滑的市场打败；20年来我见证了黑客的梦想、汗水和努力，但往往只能看着

像IBM这种又老又坏或者像微软这种又新又坏的家伙们拿走了现实世界的奖励；20年来我们生活在隔离区——它还算舒服，充满着有趣的朋友，但却仍然被一个巨大的不可触摸的由主流偏见组成的屏障包围，上面写着几个大字："里面都是怪人"（"ONLY FLAKES LIVE HERE"）。

网景的声明打破了这个屏障，哪怕只有这一刻，自鸣得意的商业世界也会被黑客们的能力震撼。但人们有着强大的懒惰惯性，如果网景失败了，也许即便成功了，仍可能被看做是一个不值得重复尝试的"一次性"试验。我们将重新回到隔离区，而围墙可能会更高。

为了防止这点，我们需要网景成功。我仔细思考了我在集市模式开发中学到的东西，并给网景打电话，表示可以帮助开发他们的许可证，并策划详细的战略。1998年2月初，我应邀飞到山景城，和网景总部各种小组召开了长达7小时的会议，帮助他们设计了后来称为Mozilla公开许可证和Mozilla组织的大体轮廓。

在那里，我会见了一些硅谷和Linux全球社区的关键人物，和我交谈的每个人都清楚，帮助网景是近期优先考虑的事，网景源码发布之后，需要有一些长期的策略，现在该是谋划的时候了。

5.5 “开源”的起源

从大体上看，我们的策略很清楚，就是使用我在“大教堂与集市”中倡导的实用方法，将它们进一步完善，并有力地向公众推广。既然网景已经有兴趣说服投资者相信这个策略并不疯狂，我们也可以藉此宣传。而且，我们早已争取到了Tim O'Reilly（通过Tim，可以让整个O'Reilly&Associates站在我们这边）。

然而，真正在概念上的突破，是我们终于承认自己需要有效的市场营销，以及相应的营销技术（讲故事，形象建设，品牌再造）。

1998年2月3日，在位于山景城的VA Research（现在的VA Linux System）办公室中举行的一次会议上，“开放源码”一词由开源运动的首批参与者们发明出来，并随之成立了后来被命名为“开放源码促进会”（Open Source Initiative）的组织。

回顾过去，可以很清楚看到，“自由软件”这个术语在过去多年里给我们的活动带来了严重的危害，部分原因是“自由”一词在英语中有两个截然不同的含义，一个是指免费，一个是指不受限制。Richard Stallman（自由软件基金会的创始人）一直在捍卫“自由”这个词，尽管他常对人们说“Think free speech,not free beer”，但这个词的模糊性还是产生了严重问题，尤其是大多数自由软件目前都免费的情况下。

然而更糟糕的危害是，“自由软件”容易让人联系到对知识产权的敌意以及其他一些很难让MIS管理者喜欢的东西。

我们知道在这里辩论自由软件基金会的真正立场有点离题，比如说它并不是对所有知识产权都有敌意，也并不是完全意义上的大公无私。我们也认识到，网景开源的压力和FSF的实际立场并没有什么关系，我们只是觉得，FSF的传教在事实上产生了负面效果（在商业出版物和企业界的观念中，“自由软件”总是和那些负面而陈旧的观念联系在一起）。

网景事件后，我们的成功将取决于能否把FSF那一套负面的东西替换成正面而务实的说法，让管理者和投资者因为高可用性、低成本和更好的特性而乐于接受开源。

使用传统的市场术语，我们的工作就是“品牌再塑”，建立起“品牌美誉度”，让企业界迫不及待地购买我们的产品。

Linus Torvalds在会议第二天就表示了赞成。之后几天我们行动了起来，Bruce Perens注册了opensource.org域名并在一周内搭起了首版的Open Source网站（http://www.opensource.edu）。他建议把Debian Free Software Guidelines改为“开放源码定义”（Open Source Definition，http://www.opensource.org/osd.html），并尝试把“Open Source”注册成认证标识，以便我们可以合法地要求人们在符合OSD的产品上使用“Open Source”标识。

虽然还处于早期阶段，但对我来说，推进战略所需的详细战术已经非常清楚了（我们在首次会议中明确讨论了这些战术），关键点如下。

1.忘掉自底向上，开始自顶向下

看上去最明白无误的是，传统上UNIX自底向上传播“福音”（即工程师用理性方式说服老板）的策略已经失败。这种方式很天真，而且很容易会被微软打败。网景的这次突破行动采用了相反的做法：战略决策者（Jim Barksdale）拿定主意，然后向下属强制推行这个愿景。

总之，不要再自底向上，我们应该自顶向下传播“福音”，即直接说服CEO/CTO/CIO这类人。

2.Linux是我们最好的例证

我们必须大力宣扬Linux。是的，开源世界里还有其他一些不错的东西，这场运动也会向它们致敬，但Linux有着最好的知名度，有着最广泛的软件库，以及最大的开发社区。如果Linux都不能帮助突破，说实话，其他的就更指望不上了。

3.抓住财富500强

除了财富500强，市场中另有一部分也很能花钱（最明显的例子是小企业和自由职业者），但这部分市场过于分散而且很难抓住。财富500强不只是有钱，而且有集中的和相对容易获取的钱，因此软件产业在很大程度上会按照财富500强的意愿行事。所以，我们首先应该说服财富500强。

4.赢得那些效劳财富500强的有威望媒体

把目标选定为财富500强，意味着我们需要赢得那些给上层决策者和投资人营造舆论环境的媒体。特别是《纽约时报》、《华尔街日报》、《经济学人》、《福布斯》以及《巴伦周刊》等等。

从这点看来，争取技术行业刊物是必要的，但远远不够，若要席卷华尔街，一个重要和基本的条件是先鼓动起精英主流媒体。

5.说服黑客，游击市场

很明显，说服黑客社区自身与说服主流一样重要。如果只是一个或几个代表言之凿凿而大多数草根黑客并不买账，那可就差点意思了。

6.使用“Open Source”认证标识，保持纯净度

我们面临的一个威胁是：微软或其他大供应商可能会采取“拥抱并拓展”（embrace and extend）策略破坏“Open Source”一词，使它失去我们要传达的理念。所以Bruce Perens和我一开始就决定把这一术语注册成认证标识并把它和“开源定义”（Open Source Definition，也即Debian Free Software Guidelines的拷贝）绑定。这样我们可以利用法律诉讼的威慑力吓跑那些可能的滥用者。

事情发展的结局是，美国专利商标局不同意为这样一个描述性短语发放专利商标。一年后我们不得不放弃正式注册“Open Source”的努力，幸运的是，这个术语在媒体和其他地方都获得了知名度，我们所担心的那种滥用并没有（至少在2000年11月前）真正出现。

5.6 意外的革命者

策划这种战略相对比较容易，对我来说，困难的部分是我必须要接受自己无法逃避的角色。

从一开始我就明白一件事，那就是媒体完全不喜欢抽象的东西，如果没有名人站在他们面前说个什么观点，他们就不会报道它。所有东西都必须有故事、有戏剧性、有冲突、有亮点，否则的话，大多数记者宁愿去睡觉——即使他们不这样，编辑也会。

所以我知道，对于网景的这次机会，我们需要一个非常有个性的人代表社区做出响应，我们需要一个“鼓动者”、一个“代言人”、一个“宣传家”、一个“大使”、一个“传道者”——这个人一方面和CEO们秘密往来，一方面站在屋顶又唱又跳大喊大叫地吸引记者，不断刺激媒体直到这个机器的齿轮碾出一句：革命来了！

与大多数黑客不同，我有着外向的神经特质，有着丰富的和媒体打交道的经验，看看周围，我无法找到比我更有资格扮演“传道者”的人。但我并不想干这个，因为我知道这会耗费我生命中若干月或若干年的时间，我的隐私将会完蛋，我很可能会被主流媒体歪曲成为一个电脑怪人（geek），或者（更糟糕的是）被我自己部落中相当一部分人鄙视为沽名钓誉或贪图名利的家伙，而比这些全都加在一起还要糟糕的是，我很可能不再有时间继续当黑客！

我必须问自己：你是否受够了你的部落总是失去成功的机会？答案

明显是“yes”。既然如此，那就有必要投入这个吃力不讨好的工作，成为一个公众人物和媒体名人。

我在编辑《新黑客词典》（The New Hacker's Dictionary）时就已学到一些基本的媒体手法。这次我更加用心，发展出一整套媒体运用理论并加以实践，这套理论围绕我所称的“吸引人的不一致”来煽动起人们对传道者的好奇心，然后充分利用这种好奇心来推广我的理念。

这本书并不是详尽展开这套理论的地方，想象一下“最高水平的挑逗”，聪明的读者就可以推测出很多。事实上，我会在采访中兴致盎然地谈论起我对枪支、无政府主义以及巫术的兴趣，同时尽可能让我看起来体体面面、充满童心并且拥有典型美国式的阳光心态。诀窍在于，语出惊人但却传达出让人放心的诚实和单纯。（注意：若想成功使用本诀窍，你必须真的是那样，如果其中任何一点做不到，推荐你不要使用，否则极有可能让你原形毕露。）

利用“open source”这个标签，加上我对自己“传道者”身份的刻意宣传，带来了我所预料的正反两方面效果。在网景宣布开源后的十个月内，媒体关于Linux和开源世界的报道呈现出稳步的指数式增长。这段时间里，大约三分之一的文章直接引用了我所说的话，剩下三分之二也大多把我做为背景资料。与此同时，一小部分黑客则高调指责我是一个恶劣的自大者。我对这两种结果都尽可能保持一种幽默的心态（尽管有时这并不容易）。

我一开始的设想是，我最终会将传道者的角色传给一个继任者——

不管是个人还是组织。总有一天，机构威信的效果会因为其更深厚的基础而超越个人魅力（在我看来，这一天来得越早越好！）。我正尝试将我的个人关系和用心建立起来的媒体上的声誉移交给开放源码促进会，一个专门管理“Open Source”标识的非盈利组织。写下本文之际我是这个组织的主席，但我希望自己不要总在这个位子上。

5.7　运动的各个阶段

开源运动开始于山景城会议，并很快通过互联网形成了一个非正式的同盟（其中包括网景和O'Reilly&Associates公司的关键人物）。下文中的“我们”指的就是这一同盟。

从2月3日到3月31日，在网景实际发布源码的这段时间内，我们主要集中精力于让黑客社区相信“Open Source”标签以及它所引起的争论是我们说服主流的一种最佳做法。结果是，变化比我们预期的要容易。我们发现社区中有很多压抑已久的需求：他们希望有一个不像自由软件基金会那么教条的新理念。

3月7日召开的自由软件峰会（Free Software Summit）上，Tim O'Reilly邀请了20多位主要自由软件项目的带头人。他们投票采纳了“Open Source”这一名称，这意味着在开发者草根群体中已经逐渐明朗的趋势得到了正式认可。在山景城会议的6周后，社区中的大多数人都已在使用我们的语言。

自由软件峰会之后所做的宣传使得主流媒体开始了解开源，并且让人们知道并不只是网景才接受开源的概念，开源现象的影响力已经超过了任何一个互联网社区局外人的认识。开源程序远远不再是处于边缘地带的挑战者，它们已经成为提供互联网基础架构各关键元素的市场领导者。Apache是Web Server的领导者，有着超过50%的市场份额（现在已增长超过60%），Perl则是新兴网络应用的主要编程语言，Sendmail传

输了所有互联网邮件的80%，而无所不在的域名系统（它使我们可以用www.yahoo.com这种网址而不是拗口难记的IP地址）则几乎完全依靠名为BIND的开源程序。正如Tim O'Reilly在峰会之后新闻发布会上的发言，当时他指着聚集在一起的程序员和项目领导人说："这些人仅仅靠着他们的想法和网络化的开发者社区，就创造出了主导市场份额的产品。"如果大公司也采纳开源这一套方法，那会发生些什么？

我们在"空战"上已经有了好的开端，并改变了媒体的看法。但我们也需要在"陆战"上保持势头，4月，在峰会和网景实际发布源代码之后，我们的主要精力转移到动员尽可能多的人接受"开源"，目标是让网景的行动不那么孤单——这同时也是买个保险，以防网景计划执行不利而导致失败。

这是最让人担心的时刻。表面上，一切看上去都很顺利；Linux在技术上越来越强大，开源活动正在享受着行业刊物壮观的爆炸式报道，我们甚至在主流媒体也得到了正面评价。然而，我不安地意识到，我们的成功还很脆弱。Mozilla在经历一开始的代码贡献风潮之后，因为要求使用专有的Motif工具包，社区参与度急剧下降；没有任何一家大型的独立软件供应商承诺向Linux移植；网景看上去依然孤立无援，其浏览器市场占有率正因IE而不断丧失。任何稍微严重一点的负面事件都会让行动在媒体报道和公众舆论中产生令人沮丧的倒退。

5月7日，我们迎来了网景之后的第一次真正突破，Corel公司发布了基于Linux的Netwinder网络计算机。但这并不够，要继续保持这种势

头，我们不仅需要那些饥饿的二线厂商，还需要行业老大的支持。因此，直到7月中旬Oracle和Informix宣布加入，才算给这一敏感期画上了句号。

数据库方面加入Linux阵营比我预期早了3个月，但并不是太早。我们曾好奇在没有主要ISV（独立软件开发商）的支持下，正面声音能够持续多久，并着急到哪里找到这些支持。在Oracle和Informix宣布支持Linux后，其他ISV也纷纷开始宣布支持Linux。这样，即便Mozilla失败了我们也能继续下去。

7月中旬至11月初这段时间是巩固期。以《经济学人》的几篇文章和《福布斯》封面故事为开端，金融媒体（这是我最初就选定的目标）开始对我们有了相当稳定的报道。各种各样的软件和硬件供应商向开源社区做出试探，并开始制定战略以图从这种新模式中获取收益。而其中最大的闭源供应商，开始感受到严重的不安。

他们到底有多么不安，可以从恶名远扬的微软万圣节文件（http://www.opensource.org/halloween/）中一窥究竟。这些内部战略文件已经认识到开源模式的威力，并大致描绘出微软的对策：破坏开源所依赖的开放协议、垄断消费者的选择。

万圣节文件引起了轰动。Linux如果成功，微软将成为最大的输家，微软这些文件无疑是开源开发优势的有力证明。它同时也印证了很多人内心深处的怀疑，那就是微软会采取各种战术阻碍开源的发展。

万圣节文件在11月最初几个星期内引发了大量的新闻报道。它引发

了人们对开源现象的新一轮关注，这个意外发现让我们进一步坚信了这几个月所做的决策。此事直接导致美林证券邀请我和其选择的主要投资人团体就软件业现状和开源前景进行了交谈。华尔街，终于开始正视我们了。

接下来的六个月让我体验了越来越超现实的对比。一方面，我受邀给财富100强企业战略家和技术投资人作开源方面的演讲，在我生命中的第一次，我坐上了头等舱，见识了超长豪华轿车的内饰；另一方面，我和草根黑客们一起在街头示威——正如喧闹有趣的Windows Refund Day游行，那一天是1999年2月15日，旧金山湾区的一群Linux用户在媒体全程报道下前往微软办公处示威，要求微软按照Microsoft End User License对捆绑在他们机器上但并未使用的Windows退款。

那个周末我要在理性基金会（Reason Foundation）举办的会议上发言，所以那天我正好会在旧金山，于是我主动要求成为这次活动的带队。由于互联网连环漫画网站“User Friendly”在1998年12月曾诙谐地把我编排为“星球大战”剧情里（http://www.userfriendly.org/cartoons/archives/98dec/19981203.html）的主角。所以我和组织者开玩笑说要在游行那天穿上绝地武士Obi-Wan Kenobi的戏服。

让我惊讶的是，当我抵达现场的时候，发现组织者竟然真的给我做了一件还算不错的绝地武士服。于是我走在队伍前面，带领人们举着揶揄好玩的标语牌、美国国旗和一个相当大的塑料企鹅，向周围兴奋的记

者们喊着：“源码与你同在！”更让我惊讶的是，他们还推选我作为代表向媒体发表声明。

CNBC电视台对这次游行做了报道，我想我们之中没有人会真的为此感到惊讶。这次游行是一次巨大的成功，微软公关部门在还没有从万圣节文件曝光事件中恢复过来的时候，又遭到如此重重一击。几周后，一些主要的PC和笔记本制造商宣布他们将推出不预装Windows软件的机器，当然价格里也不会包含“微软税”。我们的街头游击战，看上去击中了对方的要害。

5.8　陆战进展

当开源运动在媒体“空战”上打得如火如荼的时候，在关键技术和市场活动这些“陆战”上也取得了进展。我将简要回顾其中部分内容，因为这些事件很有意思地把媒体导向和大众认知结合了起来。

在网景发布源码后的18个月内，Linux的能力得到了快速成长。它对SMP（对称多处理器）的可靠支持以及向64位移植的实际完成也给未来战斗打下了坚实基础。

给《泰坦尼克号》电影渲染场景的Linux盒子堆满了整个屋子，这着实让那些提供昂贵图形引擎的制造商们感到害怕。Beowulf项目（通过廉价机器搭建起超级计算机）向人们表明Linux所具备的集群能力，能够成功应用在甚至是最尖端的科学计算上。

Linux的开源对手们并没有获得任何引人注目的成就，专有UNIX系统仍在不断失去市场份额；到年中时，事实上只有Linux和NT系统仍在实际获取财富500强的市场份额，而到深秋时，Linux获取市场份额的增速更快了（并且抢走NT的市场份额要多于其他UNIX的）。

在保持领先的同时，Apache在Web服务器市场的份额仍然不断增长（到1999年8月，Apache及其衍生品已经运行在61%的全球公众可接入Web服务器上）。1998年11月，网景浏览器已经扭转其市场份额下滑局面，并开始从IE那里夺回部分市场。

1999年4月，著名的计算机市场研究组织IDG预测Linux在2003年的增

速将会是其他所有服务器操作系统增速的两倍，也即比Windows NT更快。5月，Kleiner-Perkins（硅谷领先的风险投资公司）首先开始为基于Linux的创业公司提供融资。

唯一的负面消息是Mozilla项目不断出现问题。这些问题在Jamie Zawinski离职时爆发出来，作为Mozilla的共同创始人和Mozilla项目的公众面孔，Zawinski在源代码发布一年零一天后辞职，并抱怨糟糕的项目管理导致很多机会丧失。

但Mozilla的麻烦并没有明显减缓公众认同开源的步伐，这无疑表明开源此时已经拥有了巨大的动能。行业刊物很明显对开源也有了更正确的认识，正如Zawinski那句名言所说的："开源（很伟大，但它）并不能点石成金。"

1999年初在各大独立软件开发商之间掀起了一股风潮，效仿之前主要数据库厂商的做法，他们纷纷将其商业应用移植到Linux上。到7月下旬，其中最大一家公司，Computer Associates，宣布其大部分产品线都将支持Linux。1999年8月针对2000名IT管理者进行调查的初步结果显示，49%的受访者都认为Linux在其企业级计算战略中是"重要或必不可少"的因素。IDC的另一项调查称Linux自1998年以来有着"令人吃惊的增长速度"（虽然市场研究表明那时Linux的使用率在统计学上仍不显著），而13%的受访者现在已经在其业务运营上使用Linux了。

1999年，Red Hat Linux、VA Linux以及其他Linux公司的成功上市引发了一波Linux IPO热潮。当那些被投资者过分高估的互联网公司在

2000年3月历经市场回调而难以为继时，这些围绕开源以盈利为目的公司明显是错不了的，他们会持续成为投资者追逐的热点。

5.9　走进未来

对前面这些内容的复述，部分原因是为了记录这段历史，更重要的是，这些内容为我们理解近期趋势和未来走向提供了背景信息。

首先，对明年（即1999年——译者注）比较有把握的预测是：

· 开源开发者人数将继续激增，其背后推力是越来越便宜的PC硬件和高速互联网连接。

· Linux将继续保持领先，其庞大规模的开发者社区在能力将压倒性超过开源BSD团体和人数极少的HURD团队，虽然后两者有着更高的平均技术水平。

· 承诺支持Linux平台的ISV将会急剧增多；数据库供应商的承诺是一个转折点。

· 开源运动将乘胜追击，并在CEO/CTO/CIO及投资者人群中的认可度大幅提升。在是否使用开源产品上，MIS管理者会感受到不断增加的压力，压力并非来自下属，而是来自上层管理者。

· Samba-over-Linux的暗中部署将会替换掉越来越多的NT机器，即使在那些全盘实行微软策略的场所也是如此。

· 专有UNIX的市场占有率将继续受到侵蚀。至少有一个较弱竞争者（比如DG-UX或HP-UX）会走下历史舞台，那时，分析师会将其归功于Linux，而不是微软。

· 微软不会拥有企业级的操作系统，因为Windows 2000将不会以可

用形式发行。（已有6000万行而且还在不断增长的代码，会导致其开发失控。）

上面这些预测是我在1998年12月中旬写下的。2000年11月，两年过去了，这些预测仍然有效。只是最后一条存在争议：为了发行Winodws 2000，微软大幅度缩减了其特性列表，但市场占有率并未如其所愿。

对这些趋势进行推理，我们能得到略微大胆一些的中期预测（未来18至32个月）。

· 对商业用户提供开源操作系统的运行支持，将会成为很好的买卖，供应商不仅能从中获益，也会推动开源在商业领域的发展。（1999年Linux-Care的推出使这一点已经实现，同年，IBM、HP以及其他一些公司也宣布提供Linux支持服务。）

· 开源操作系统（以Linux为首）将占领ISP和商业数据中心市场，NT对此将无可奈何，低成本、开源以及真正24/7的可靠性，实在让人无法拒绝。

· 专有UNIX市场将几乎完全垮台。Solaris会因为SUN的高端硬件而看上去还算安全，但大多数专有UNIX将很快成为遗留系统。（2000年初，SGI的IRIX走到了尽头，因为SGI自身都已开始正式使用Linux了。2000年中期，SCO同意Caldera对其开展收购。现在看来，一些UNIX硬件供应商将很可能没有异议地转向Linux，就像SGI做得那样好。）

· Windows 2000要么会被取消，要么一面世就玩完，无论是哪种情

况，它都将遭遇可怕的灾难，成为微软有史以来最糟糕的重大失败。然而，由于其纯熟的市场公关能力，在未来2年内，微软对消费者桌面的控制能力几乎不会受到影响。

（2000年中期，IDG新发布的调查认为Windows 2000“一面世就玩完”更可能发生，大多数大公司受访者完全不愿意部署其最初发布版本。而已经部属的公司，则经历了严重的安全性和稳定性问题。即便微软自身在2000年10月初和11月末两次被黑客攻入，也没有促使其提高产品的安全性。）

乍看上去，这些趋势都表明Linux将成为最后的胜利者。但事实上没有那么简单，微软通过桌面市场已经获取如此巨量的钱财和市场影响力，它可没有那么容易退场，即便Windows 2000有一火车的问题。

但我们仍有理由相信微软在2001年会有大麻烦，这些问题与Linux无关，也与司法部无关。微软59%的收入来自于向计算机OEM厂商销售固定价格的预安装许可费，随着硬件价格下降，这部分收入将面临压力。这些固定许可费在OEM总体利润中的占比越来越多，在某个临界点，OEM将不得不从Redmond巨人（指微软）那里追讨盈利。从家电市场以及PDA市场我们可以推算出这个临界点：大约是零售价350美元。根据之前的趋势，台式机价格将在2001年年中之前跌破350美元，到那时，为了求生存，OEM厂商将不得不背叛微软阵营。

即便微软采取这样的应对策略也无济于事：通过收取机器零售价格的百分比来取代对每台机器收取固定价格。而OEM也有简单的办法应

对——对那些昂贵的外设（如显示器）另行计价，即便不这样做，华尔街也会把这种策略视作微软已经对未来收入失去控制的表现。不管怎样，微软的收入看上去很可能早在司法部最终裁决到来之前就会崩溃。

两年之后的水晶球显得有点模糊，我们会有怎样的未来取决于这样的问题：司法部能否成功让微软分裂？BeOS、OS/2、Mac OS/X或其他生态的闭源OS，或全新设计的OS，它们能否走向开放并和有着30年设计基础的Linux展开有效竞争？至少Y2K是虎头蛇尾地结束了……

这些几乎都是无法预测的。但有一个问题值得思量：Linux社区能否给整个系统提供一个终端用户友好的GUI界面？

1999年本书第一版问世时，我预测2000年末/2001年初最有可能的情景是Linux有效控制了服务器、数据中心、ISP以及互联网，而微软仍然保持对桌面的控制。到2000年11月，除了在大公司的数据中心，这一预言得到了相当全面的证实，而缺失的这一块也很有可能在数月之内补上。

今后的发展取决于GNOME、KDE或其他一些基于Linux的GUI（以及基于它构建或重构的应用程序）能否好到足以挑战微软的主场地位。

如果这主要是技术问题的话，其结果就不难推断了。但它并不是，问题主要出在人机工程学设计和界面心理学上，黑客在这些方面从来就很弱。黑客在为其他黑客设计接口时做得很好，但他们往往不善于为另外95%人群的思维过程建模，从而写不出能让终端用户J.Random和他姨妈Tillie愿意花钱购买的界面。

1999年的问题是“应用”。现在看来我们已经改变了足够多的ISV的立场，使我们得到了很多并非自己写的“应用”。我相信2001年及以后的问题是我们能否发展到这样一种境界：满足（并超越！）Macintosh所设立的界面质量标准，同时又保留传统UNIX的优点。

2000年年中传来的消息是，我们可能会得到来自Macintosh发明者的帮助！Andy Hertzfeld和Macintosh最初设计团队的部分成员已经组建了一家开源公司Eazel，其明确目标就是把Macintosh的魔力带给Linux。

我们半开玩笑地说要“统治世界”，但做到这点的唯一途径就是服务世界。这意味着要服务终端用户J.Random和他的姨妈Tillie，意味着要学会如何思考让我们以全新的方式来做事，意味着要坚决将用户环境的复杂性减少到最低程度。

电脑是人类的工具。因此，硬件设计和软件设计最终要回归到为人类设计，而且是为全人类设计。

这条路是漫长的，而且并不容易走。但我认为，黑客社区联合企业界的新朋友，必将完成这一任务。就像Obi-Wan Kenobi所说的那样，“源码将与我们同在”。

[1] 原文为shot heard'round the world，最早出现在Ralph Waldo Emerson的诗歌“Concord Hymn”（1837）中，是指美国独立战争的开端，也即在Lexington和Concord的武装冲突，该冲突以及随后的一连串事件导致了独立宣言的签署和十三个殖民地的独立。这个短语现在多用来指代历史性事件。——译者注

[2]　模因(meme)是由新达尔文主义（Neo-Darwinism）倡导者Richard Dawkins在他1976年所著《自私的基因》（The Selfish Gene)中首次提出的。模因被定义为文化的基本单位，它通过被模仿和被复制而得到传递。模因有时被比喻为“病毒”，它可以感染其他人的大脑或者传染到其他人的大脑中，而一个人一旦被感染，“病毒”就会寄生在他的头脑中，并将其传播给其他人。模因论是解释文化进化规律的新理论，它借用生物进化模式探讨模因的复制、传播和进化，对文化具有传承性这种本质特征的进化规律进行诠释。模因工程则是基于模因论创造和发展某种理论的过程。——译者注

[3] Thomas Paine（1737－1809），英裔美国政治活动家、作家、政治理论家和革命家，被广泛认为是美国的开国元勋之一。他撰写了若干富有影响力的小册子，其中最有名的是《常识》，该文在美国独立战争中起到了鼓动人心和激发斗志的作用。他是自由主义和宪政共和政府的倡导者，在《人的权利》一书中他概括总结了自己的政治哲学。——译者注

后记

软件之外

本书这几篇文章仅仅是个开端而非定论，开源软件还有太多的问题没有解决。开源现象引发了其他创造性工作领域以及知识产权相关的许多问题，但未能真正给出较好的答案。

经常有人问我这样的问题：开源模式能否有效应用于软件之外的其他商品？最常问到的包括音乐、书籍、计算机及电子类硬件的设计。还经常有人问我：你是否认为开源模式蕴含着政治意味？

在音乐、书籍、硬件以及政治方面，我有不少自己的观点，其中一些也确实和本书中提到的同行评审、去中心化（decentralization）以及开放性相关。欢迎有兴趣的读者访问我的主页 http://www.tuxedo.org/~esr/并写下你的观点。然而，作为开源的大使和理论研究者，在谈及开源时，我通常有意回避这样的推断。

道理很简单：仗要一场一场地打。我的部落现在要打的仗，是提高软件消费者对软件质量和可靠性的期望，并颠覆软件产业现行的标准做法。

我们面对的是对手用巨额资金、品牌优势以及垄断力量构筑起的重重防线，这一仗不好打，但从逻辑和经济学上看，我们能赢，而且必然会赢，我们需要专注在既定目标上。

专注就意味着不要游走到其他有诱惑性的分支上，这一点是我经常觉得需要向黑客们强调的，因为我们过去的代表们表现出了强烈的意识形态化倾向，他们原本可以把精力集中在相对更有效和更务实的内容上。

是的，开源的成功确实引发了人们对命令控制系统（command-and-control system）、不透明性、集中式管理（centralization）以及某类知识产权的怀疑，不得不说，在如何对待个体之间、机构之间以及个体与机构之间的关系上，它向世人建议了一种广义上的自由主义观念（或至少是与自由主义相协调的）。

虽然话已至此，但我觉得在目前情况下，避免过度引申这些观点是更合适的做法。举个比较恰当的例子，音乐以及多数书籍不同于软件，它们通常不需要除错和维护，同行评审的效用会因此降低很多，选择开源的理性动机也会几近消失。我不想因开源在其他领域的可能失败而削弱开源在软件上一定会赢的说服力。

我希望开源运动能在三到五年内（2003~2005年）赢得软件上的胜利。当这个愿望实现时，当其成果显现时，它们将会融入非程序员的背景文化。到那时，将这些洞见推广到更广阔的领域内会更合适。

那时，即便我们黑客不去谈什么意识形态，我们一样会改变这个世界。

附录A
如何成为一名黑客

A.1　为什么写下此文

作为“黑客行话”（Jargon File，http://www.tuxedo.org/jargon/）以及其他几篇流传较广的类似文章的作者，我经常收到热情的网络新手的邮件，他们会问：“我怎样才能成为黑客高手？”相当奇怪的是，我没有看到任何FAQ或者网络文章谈及这个重要问题，所以我来写一篇。

如果你读的是离线版本，你可以在这个网址找到它的最新在线版本：http://www.tuxedo.org/~esr/faqs/hacker-howto.html。

注意：本文最后有一个FAQ（常见问题解答），在向我发邮件提问之前请先读它两遍。

A.2　什么是黑客

在“黑客行话”这篇文章里，有一堆关于“黑客”这个术语的定义，大多数都涉及“技术高超”、“热衷于解决问题”和“突破极限”这样的特点，如果你想知道如何成为一名黑客，其实真正重要的只有两点。

长期以来，存在一个崇尚共享文化且成员都是编程专家和网络高手的社区，其历史可以追溯到数十年前第一台分时小型机诞生和ARPAnet还处于最早实验期的年代。这个社区的成员创造了“黑客”一词，黑客构建了互联网，黑客造就了现如今的UNIX操作系统，黑客运转起了Usenet，黑客让WWW发挥作用。如果你是这个文化的一部分，如果你对这个社区有贡献，社区中其他人知道你并称你为黑客，那你就是一名黑客。

黑客精神并不局限在软件文化中。人们会把黑客态度用在其他事情上，比如电子或音乐。事实上，对任何科学和艺术，在其最高水平的活动中都可以发现黑客精神。软件黑客若是识别出其他领域的同道中人，也会称他们为“黑客”。所以有人说黑客的天性其实是独立于他们所从事工作的。在本文余下部分中，我将着重讨论软件黑客的技术、态度及其文化传统，正是这一文化产生了“黑客”一词。

有一群人高调声称自己是黑客，但他们并不是。这些人（大多是些毛头小子）的主要目的是攻入他人计算机或者破解电话系统。真正的黑客称他们为“骇客”（cracker），而且完全不想搭理他们。多数真正的黑

客认为，骇客懒惰，缺乏责任感，而且不是很聪明，如果你的目标是能够攻破安全系统，那不会让你成为黑客，就好比学会热线（hotwire）发动汽车并不会让你成为汽车工程师一样。遗憾的是，很多记者和作者错误地使用“黑客”一词来描述骇客，这使得真正的黑客极为不满。

两者最根本的区别是：黑客搞建设，骇客搞破坏。

如果你想成为一名黑客，请接着读下去。如果你想成为一名骇客，去读alt.2600新闻组（news:alt.2600），如果你发现自己并没有想象中那么聪明，做好准备去蹲上5到10次监狱吧。这就是我想对骇客说的。

A.3　黑客的态度

黑客解决问题并做出东西，他们相信自由，并自愿地互相帮助。要想被别人认可是一名黑客，你的行为必须要表现出你拥有这种态度。当然，如果要做到这点，你必须要真的信奉这种态度。

如果你把培养黑客态度当做是获取黑客文化接受的途径，那可就差远了。你要打心眼里认为这些态度对你至关重要——这会帮助你学习并保持热情。就像所有其他创造性活动一样，要想成为大师，最有效的方法是模仿大师的思维模式——不仅在理智上，还要在情感上。

所以，如果你想成为黑客，重复下面这些事，直到你信奉它们。

1.这个世界充满了迷人的问题等待人们去解决。

做一名黑客有很多乐趣，但这是一种需要努力才能获得的乐趣。而努力需要动力，成功运动员的动力来自于控制自己身体和超越自己过往生理极限的愉悦。类似地，成为一名黑客，你必须要对解决问题、磨砺技能和智力挑战有着基本的兴奋感。

如果你不是那种天生对此就很有感觉的人，你需要把自己变成这种人，否则你会发现你做黑客的能量会被性、金钱以及社会认可这类让你分心的东西慢慢耗尽。

（你还必须要培养出一种对自己学习能力的信心——你要相信，即便你没有掌握解决某个问题所需的全部知识，如果你成功处理了其中一小部分而且从中学到东西，你将会学到足够多的知识去解决下一小部分

——如此往复，最终你会解决整个问题。）

2.不要解决一个问题两次。

创造性头脑是无比珍贵的有限资源，它们不应浪费在重新发明轮子这种事上，尤其是还有这么多迷人的新问题在那里等着的时候。

想要像一名黑客，你必须要相信：其他黑客的思考时间是很宝贵的——它是如此宝贵，以至于共享信息、解决问题并将解决办法馈赠给其他黑客几乎就是你的道德义务，这样，其他黑客就可以去解决新问题，而不是永无休止地去重复解决老问题。

（你不必觉得你有义务把所有创造性产品都贡献出来，尽管这样做的黑客能获得其他黑客最大的尊重。出售软件以换取食物、房租和计算机并不违反黑客价值观，运用你的黑客技能来养家甚至发大财也都没什么，只要你在做这些的时候，不要忘记你对理想的忠诚以及你的黑客朋友就行。）

3.无聊和乏味是有害的。

黑客（以及有创造性的人们）应该从来不会觉得无聊，也不会去做那些乏味而愚蠢的重复性工作，如果这种事情发生，意味着他们没有做只有他们才能做的事——解决新问题。这种浪费会伤害到每个人，无聊和乏味不仅仅是不好，而且是有害的。

要当一名黑客，你必须得非常相信这点，并希望尽可能将那些无趣的事情自动化，这不仅是为自己，也是为其他人（尤其是其他黑客）。

关于这点，有一个明显的例外：黑客有时会做一些外人看上去重复

或无趣的事，其实他们这样做只是为了清空大脑，或是为了获得某种技能，或是为了获取某些在其他情况下无法获取的经验。注意这是他们自愿的——任何有想法的人都不该被强迫去做那些他们觉得无聊的事。）

4.自由是好事。

黑客天生是反权威的。如果有人能命令你，他就能让你做不成你特别想做的事——而且，如果探究权威者的思维，你会发现其理由往往愚蠢得令人发指。所以无论什么地方出现权威主义倾向，你都要与之抗争，以免他们压迫你和其他黑客。

（这并不是说要和所有权力抗争。儿童需要成人的指导，罪犯需要强制关押。黑客应该接受某种类型的权力，虽然他在服从命令上要花些时间，但换来了更多他想要的东西。这是一种有节制的、理性的谈判，而绝不是当权者所想要的那种个人顺从。）

滥用权威者靠审查和保密而强大，他们不信任自愿合作和信息共享——他们只喜欢他们控制之下的“合作”。若要做得像一名黑客，对审查、保密以及使用武力或欺骗这类行为，你必须要有一种直觉上的反感，而且你必须愿意与之抗争。

5.态度不能代替能力。

要想成为黑客，你必须养成这些态度。但只凭态度并不会让你成为黑客，就像只凭态度不会让你成为冠军运动员或摇滚明星一样。要成为一名黑客，你需要智慧、实践、投入和努力。

所以，你必须学会持怀疑态度并尊敬每种能力。黑客不会让装腔作

势者浪费他们的时间，黑客崇拜能力——特别是黑客能力，但并不限于此，在任何事情上的能力都是好的。那种只有极少数人才能驾驭的技术能力尤其好，而那种需要思维敏锐、动手能力强和全神贯注的技术能力最好。

如果你崇尚能力，你会从动手开发中获得乐趣——辛苦的工作和投入将成为紧张的比赛而不是苦工。要想成长为黑客，这至关重要。

A.4　黑客的基本技能

对黑客来说，态度固然重要，但技能更重要。态度并不是能力的替代品，有一些特定的基本技能是你必须要掌握的，否则没有黑客愿意称你为“黑客”。

这些基本技能会随时间缓慢变化，因为技术进展会产生新技能并淘汰旧技能。例如，以前它会包括机器语言编程，而不会有现在的HTML。不过目前看来，很明显基本技能包括以下几个方面。

1.学习如何编程。

这当然是一项基本的黑客技能。如果你不懂任何计算机语言，我建议从Python入门。它设计整洁，文档良好，对初学者很友好。作为一个好的入门语言的同时，它并不是一个玩具，它非常强大、灵活，完全适用于大型项目，对此我曾写过一个较为详细的评估（http://noframes.linuxjournal.com/lj-issues/issue73/3882.html）。其教程则可在Python网站上找到（http://www.python.org）。

Java也是一个不错的学习编程的语言。它比Python要难，但可以产生比Python更快的代码，我想它可以作为一个优秀的第二语言。

但是要注意，如果你只懂一两种语言，那你不会达到黑客的技能水平，甚至连程序员都称不上——你需要学会以一种独立于任何语言的一般方式来思考编程问题。要成为一名真正的黑客，你需要到达这样的程度：你能把手册上的东西和你已经知道的东西联系起来，从而可以在几

天之内学会一门新语言。这意味着你需要学习几种非常不同的语言。

如果你想进入真正的编程领域，则必须学习C，这是UNIX的核心语言。C++和C的关系很密切；如果你懂其中一个，学另一个并不会太难。不过，这两种语言都不是好的入门语言。

其他对黑客而言比较重要的语言有Perl(http://www.perl.com)和LISP（http://snaefell.tamu.edu/~colin/lp/)。Perl由于其实用性而值得一学，它广泛应用于动态网页和系统管理，因此即便你永远不写Perl，你也应该学习如何读它。LISP也很值得学习，当你最终掌握它时，你会得到深刻而彻悟的体验，这会让你在未来的日子里成为更好的程序员，即便在实际上你不怎么用它。

其实，最好把这五种语言（Python、Java、C/C++、Perl和LISP）都学了。它们不只是最重要的黑客语言，还代表了截然不同的编程方法，每一种都会让你受益匪浅。

这里我没法给出如何学习编程的完整指导——这是个复杂的技能。但我可以告诉你，书本和课程也无法做到（很多黑客，也许是大多数黑客，都是自学成才的）。你可以从书本上学到语言特征，但那只是皮毛，想要获得将知识转化为技能的思维模式，只能通过实践和跟随大师。做法就是读代码和写代码。

学习编程就像学习如何写好自然语言。最好的方式是阅读大师们写的东西，然后写你自己的东西，多读一些，多写一些，再多读些，再多写些……如此循环往复，直到你写的东西开始发展出你在经典中所体会

到的力量和简洁。

以前想要找些好代码来读挺不容易，因为几乎没有大型项目会以源码形式提供给成长中的黑客阅读和练手。现在这种情况已经发生了戏剧性的变化。开源软件、开源编程工具、开源操作系统（这些都是黑客写的）已经随处可见。这就带来了下一个话题……

2.找一个开源UNIX，学习使用和运行它。

假设你有台PC或者说有台PC可以让你用（现在的孩子们真幸福:-)）。对任何新手来说，获取黑客技能最重要的一步是搞一份Linux或是BSD-UNIX，把它装到个人机器上，然后运行它。

是的，除了UNIX，这个世上还有其他操作系统，但都是以二进制发布的——你没有代码可读，没有代码可改。尝试在DOS或Windows或MacOS之下学习黑客技能就像全身打着石膏学跳舞。

而且，UNIX是互联网的操作系统。不懂UNIX，你也可以学习使用互联网，但你不会成为互联网上的黑客。因此，相当大程度上，今天的黑客文化是以UNIX为中心的。（并不是历来如此，一些老资格黑客仍然对此不满，但UNIX和互联网之间的共生关系已经如此紧密，即便以微软的能量，也不可能真正撼动它。）

所以，找一个UNIX——我个人喜欢Linux，但还可以有别的（是的，你可以在同一台机器上既运行Linux又运行DOS/Windows）。学习它、运行它、捣鼓它、用它和互联网对话、读它的代码、改它的代码。你会得到比微软操作系统下不知好多少倍的编程工具（包括C、LISP、

Python和Perl），你会乐在其中，你吸收的知识会比你当时意识到的更多，当你成为一名黑客高手后，你就明白了。

关于如何学习UNIX的更多知识，参见Loginataka（http://www.tuxedo.org/~esr/faqs/loginataka.html），关于如何着手实践Linux，参见"Where can I get Linux"（http://linuxresources.com/apps/ftp.html）。在http://www.bsd.org可以找到BSD UNIX的帮助和资源。

（注意：如果你是个新手，我真的不建议你一个人玩Linux或BSD。对于Linux，找一个当地的Linux用户组并寻求帮助，或者可以找LISC（Linux Internet Support Co-Operative，http://www.linpeople.org)，LISC维护着IRC频道[http://openprojects.nu/services/irc.html]，你可以从那儿获得帮助。）

3.学习如何使用WWW和写HTML。

黑客文化制造的大多数东西都是看不见的，这些东西帮助工厂、办公室和大学运转，但对非黑客人群的生活没有明显影响。Web是个很大的例外，连政治家们都承认，这个庞大而闪闪发光的黑客玩具正在改变着世界。单单是这个原因（当然还有很多其他好处）你也要学习怎么掌握Web。

这并不只是说让你去学习如何使用浏览器（谁都会做这个），而是去学习怎样使用HTML这一Web标记语言。如果你不懂怎样编程，写HTML能教会你一些思维习惯，而这会有利于你学习编程。所以，先写

个主页吧。

但有一个主页，和让你成为一名黑客相去甚远。网上到处都是主页，但大多数是毫无意义和毫无内容的泥巴——它们华丽而俗气，试图吸引你的注意力，但泥巴终归是泥巴（更多内容可见HTML Hell Page:http://www.tuxedo.org/~esr/html-hell.html）。

要想有价值，你的网页必须要有内容——必须有趣并且/或者对其他黑客有用，这给我们带来了下一个话题……

A.5　地位之于黑客文化

像大多数不涉及金钱的文化一样，黑客文化靠声誉运转。你在尝试解决一些有趣的问题，但问题到底多有趣，你的方案有多好，是由你的技术同行或上司来判断的。

相应地，在黑客游戏中，你要知道你的得分主要来自于其他黑客对你技术的评价（这就是为什么只有当其他黑客都称你为黑客时你才是黑客）。这一事实之所以比较隐蔽，是因为黑客总给人以独立工作的形象，另外则缘于黑客文化的一个禁忌（虽然慢慢有所减弱但仍然很强大）：黑客认为一个人的做事动机中，不该掺杂“自我”或“外部评价”因素。

具体而言，黑客文化是人类学家所称的礼物文化。你之所以获得地位和荣誉，不是通过支配别人，不是通过美貌，也不是通过拥有别人想要的东西，而更多是通过给出。特别是给出你的时间、给出你的创造力、给出体现你技能的成果。

基本上讲，做以下五件事，会让你得到其他黑客的尊敬：

1.写开源软件。

第一件事（最核心的和最传统的）是写出其他黑客认为有趣或有用的程序，然后将程序源码发布给整个黑客文化。

（以前我们称这些作品为“free software”，但这困惑了太多的人，人们不能确定“free”究竟是想说什么。现在，我们之中很多人更愿意称之

为“开源软件”，http://www.opensource.org/。）

在黑客圈中，最受尊敬的偶像是这样一类人：他们写出了大型的、能满足广泛需求的程序，并将程序贡献了出来，使得任何人都可以使用这些程序。

2.协助测试和调试开源软件。

黑客还尊敬那些调试开源软件的人，在这个不完美的世界里，我们在软件开发过程中，不可避免地要将大量时间花费在调试阶段。这就是为什么任何开源作者稍加思考后都会告诉你，好的beta测试员无比珍贵（他们会清楚地描述症状，很好地定位问题，忍受早期版本中的bug，并愿意使用一些简单的诊断例程）。对有的人来说，调试过程可能是一场旷日持久、辛苦不堪的噩梦，而对于好的测试人员来说，可能只是一个有益于程序的清理过程罢了。

如果你是一个新手，试着去找一个正处于开发状态并且你感兴趣的程序，并试着去做一个好的beta测试员。从帮助测试到帮助排错，再到帮助修改，这是一个很自然的过程，你会从中学到很多，而且，善有善报，以后也会有人乐意帮助你。

3.发布有用的信息。

另一件好事是收集、过滤那些有用并且有趣的信息，将他们放到网页或者类似FAQ（常见问题）列表的文档中，并让人们容易看到。

技术性FAQ的维护人员甚至会得到和开源作者一样的尊敬。

4.帮助做一些基础工作。

黑客文化（以及互联网的发展）是靠志愿者推动的。有很多必要但并不吸引人的工作要有人来做——管理邮件列表，主持新闻组，维护大型软件库，提出RFC和其他技术标准等等。

把这类工作做好的人会得到很多尊敬，因为每个人都知道这些工作会耗费大量时间，并且不像玩代码那样有趣，做这些事体现了奉献精神。

5.服务黑客文化自身。

最后，你可以服务和宣传黑客文化自身，比如，写一本关于“如何成为黑客”的精准的入门教程:-)。这并不需要你在这个圈子里呆很久并且因为以上四件事中某件而成名后才能做。

毋庸置疑，黑客文化没有领导人。但它的确有文化英雄、部落长老、史学家和发言人。如果你在这个战壕里时间足够长，你可能也会成为其中之一。记住：黑客并不信任部落老人们的自我炫耀，公然追求这种名声是危险的。与其为此奋争，倒不如摆正位置，静待名声降临，然后对你的地位保持谦逊和优雅。

A.6　黑客和书呆子的关系

与普遍的认知相反，并不是只有书呆子（nerd）才能成为黑客，虽然这的确有帮助，而且有很多黑客确实是书呆子。做一个不食人间烟火的人有助于你将精力集中在真正重要的事情上，比如思考和编程。

因为这个原因，很多黑客都接受了“书呆子”这个标签，甚至会引以为荣地使用“怪人”（geek）这个更刺耳的称呼——这是他们宣告自己独立于正常社会期望的一种方式。更多深入讨论请参见对Geek的介绍(http://samsara.circus.com/~omni/geek.html)。

如果你能在集中精力做好黑客的同时还能有正常的生活，那很好，现在做到这点要比20世纪70年代我还是个新手的时候容易多了；主流社会如今对“技术呆”（techno-nerd）也友善多了，甚至有越来越多人认为黑客往往是高质量恋人和配偶的人选。

如果你是因为没有生活而被编程吸引，那也没什么——至少你不会茫然无措。然后，也许你会因此而找到生活。

A.7 格调问题

再说一次，要成为一名黑客，你必须要进入黑客的精神境地。有些事即便没有计算机也能做，虽然它们不能替代编程（没有什么能替代），但很多黑客都这么做，并能感到这些事在本质上和编程相关联。

· 学习很好地使用你的母语写作。尽管有一种陈旧的看法认为程序员写作不行，但很多黑客（包括我所知道的那些最优秀的黑客们）写得一手好文章，而且其数量多得让人吃惊。

· 阅读科幻小说，参加科幻小说集会（这是认识黑客和潜在黑客的好方法）。

· 研习禅修，并且/或者学习武术。（心智训练似乎在很多重要方面都是类似的。）

· 培养起对音乐的鉴赏力。学会欣赏一些独特类型的音乐。学习很好地演奏乐器，或者学习如何唱歌。

· 培养出你对双关语和俏皮话的欣赏能力。

以上这些事，你已经做到得越多，你就越可能是天生的黑客材料。至于为什么偏偏是这些事，并不是很清楚，但这些事与左右脑技能的结合有关，这可能是问题的关键所在（黑客不仅需要有很强的逻辑推理能力，而且要能够很快从问题的具体逻辑中跳脱出来）。

最后，不要做这些事：

· 不要使用愚蠢、浮夸的用户名或昵称。

· 不要卷入Usenet（或其他任何地方）的网络骂战。

· 不要自称为“电脑朋克”[1]，也不要浪费时间交往那些自称电脑朋克的人。

· 不要让你的贴子或email充满错误的拼写和糟糕的语法。

做上面这些事只会给你带来嘲讽。黑客们的记忆力可不差——你将会花费数年时间来让他们忘记你早年误打误撞时犯下的错误。

关于网名的问题值得着重说一下。将真实身份隐藏在虚假名字后面是骇客、warez d00dz以及其他不入流家伙们幼稚而愚蠢的行为。黑客不会这么做，黑客以自己的作为为荣，并愿意把自己的作品与真名相联系。所以，如果你有假名的话，扔掉它。在黑客文化中，假名是失败者的标识。

A.8　其他资源

Peter Seebach维护着一个优秀的黑客FAQ（http://www.plethora.net/~seebs/faqs/hacker.html），用来帮助那些不懂得如何与黑客相处的管理者。我写的“黑客圈简史”（http://www.tuxedo.org/~esr/writings/hacker-history/hacker-history.html）和“大教堂与集市”(http://www.tuxedo.org/~esr/writings/cathedral-bazaar/index.html)，对Linux开发和开源文化如何运转做了阐述，在“开垦心智层”（http://www.tuxedo.org/~esr/writings/homesteading/）中则对此话题做了更直接的探讨。

A.9　常见问题解答（FAQ）

你能教我怎么做黑客吗？

自这份文档首次发布以来，我每周都会收到一些请求（经常是一天好几次），说“请教给我关于黑客的一切”。很遗憾，我没有时间或精力做这个。我有自己的黑客项目，作为一个开放源代码倡导者，我要四处奔波，这些已经占用了我110%的时间。

即便我可以教你，从黑客的态度和技能上讲，基本上也应该是你自己教自己。你会发现，当有真正黑客愿意帮你的时候，如果你祈求他们一勺一勺地“喂”你，他们不会尊重你的。

首先，你要学一些东西。展示出你在努力尝试，展示出你有能力自学。然后再带着特定问题去请教你所遇到的黑客。

我该如何开始？

对你而言最佳的入门方式也许是参加LUG(Linux用户组)聚会。你可以在LDP Linux通用信息页面（http://MetaLab.unc.edu/LDP/intro.html）上找到这样的用户组；很可能你身边就有一个，而且多半是和某个大学或学院相关联的。如果你要求，LUG成员很可能会给你一套Linux，而且通常会帮你安装，带你入门。

我应该从什么时候开始学？现在是不是太迟了？

只要你有动力开始，任何时候都是好时候。大多数人是在15到20岁之间开始对此感兴趣的，但我知道有不少例外，高于或低于此年龄段的

都有。

我要多久才能学会黑客技能？

这取决于你的天赋和努力程度。如果足够专注，大多数人能在18个月到两年时间内掌握一套相当体面的技能。但是不要觉得这就可以了，如果你是一名真正的黑客，你会用你的一生来学习和完善你的技能。

Visual Basic或者Delphi是好的入门语言吗？

不是，因为这些语言不可移植。而且也没有这些语言的开源平台实现，因此你只能在那些厂商所支持的平台上编程。黑客可不会接受这种垄断。

Visual Basic尤其糟糕。仅从它是微软的专有语言这一点就足可以抛弃它了，而且和其他各种Basic语言一样，它的设计也很差，这会让你养成一些坏的编程习惯。

其中一个坏习惯是让你依赖单一厂商的函数库、控件和开发工具。一般而言，任何语言，若是不能得到至少Linux或某种BSD的支持，以及/或者不能得到至少三家厂商的操作系统的支持，都不值得想当黑客的你学习。

你能帮我攻破一个系统吗？或者教我如何去做？

不能。如果谁在读完这份FAQ后还问这种问题，那真是愚不可及，即便我有时间也不会教你。对这类邮件请求，我要么直接忽略，要么会不客气地回敬你。

我怎样才能拿到某人账号的密码？

这是骇客行为。请走开，傻瓜。

我被黑了。你能帮我防御攻击吗？

不能。到目前为止，问这个问题的都是使用Windows的用户。你不可能有效保护Windows使其免受攻击，它的代码和架构有太多缺陷，保护Windows就像是用漏勺从漏水的船上向外舀水。唯一可靠的防护方法就是转向Linux或其他真正安全的操作系统。

我的Windows软件出问题了，你能帮我吗？

可以。打开DOS命令行界面，输入“format c：”。你的问题会在几分钟内消失。

从哪里能找到可以对话的真正黑客？

最好的办法是在你所处的地方找一个UNIX或Linux用户组并参加他们的聚会。（你可以在Metalab的LDP站点上找到一些用户组列表的链接：http://meta-lab.unc.edu/LDP/）

（以前我说过在IRC上不会有真正的黑客，但我发觉情况有所变化，显然有些真正的黑客社区，比如GIMP和Perl，现在也有IRC频道了。）

你能推荐一些和黑客编程相关的好书吗？

我维护着一份Linux Reading List HOWTO：http://sunsite.unc.edu/LDP/HOWTO/Reading-List-HOWTO/index.html，也许它会帮到你，而且可能你会喜欢其中那篇“The Loginataka”。

我应该先学习什么语言？

HTML，如果你还不会的话。市面上有很多装帧华丽、被吹得天花

乱坠的HTML书籍，遗憾的是几乎没有一本是质量过硬的。我最喜欢的是《HTML:The Definitive Guide》（http://www.oreilly.com/catalog/html3/）

但HTML并不是一个完整的编程语言。如果你做好准备开始学习编程，我建议从Python起步，http://www.python.org。会有很多人向你推荐Perl，而且Perl现在确实比Python流行，但Perl更难学而且（在我看来）它设计得不是很好。

C的确很重要，但它比Python或者Perl都难得多。不要一开始就尝试学习C语言。

对于Windows用户，不要以为Visual Basic有多好。它会让你养成坏习惯，而且它不能跨平台，只能运行在Windows上。离它远点儿。

我需要什么样的硬件？

个人电脑曾经因其性能较弱、内存较小而给黑客的学习设置了人为限制，但时过境迁，现在任何Intel 486DX50以上配置的机器都足以胜任软件开发、X窗口运行以及互联网通信，而你能买到的最小容量的硬盘也已足够大。

选择用于学习的机器时，比较重要的一点是留心其硬件是否与Linux兼容（或者与BSD兼容，如果你选择学习BSD）。同样，如今大多机器都符合这点；唯一可能有点麻烦的是modem和打印机；有些机器含有专为Windows设计的硬件，它们无法运行Linux。

这里有最新版本的关于硬件兼容性的FAQ：

http://users.bart.nl/~patrickr/hardware-howto/Hard-ware-HOWTO.html

我是否需要憎恨并反对微软？

不需要。这并不是说微软不令人作呕，而是因为黑客文化早在微软出现之前就已存在，并会在微软成为历史之后继续存在。与其花费能量去憎恨微软，还不如去热爱你的编程手艺。写出好的代码——你就是在有力地打击微软，而且还不用担心报复。

软件开源会不会让程序员无以谋生？

不太可能，目前看来，开源软件产业非但没有使工作职位减少，反而创造了更多的工作职位。如果程序写出来比不写出来会有净经济收益，那程序员就会得到报酬，而不论程序写出来后是否免费。此外，不论写出多少“免费”软件，永远存在更多新的和定制化的应用需求。对此我在Open Source网站(http://www.open-source.org)上有更多的论述。

我该如何开始？从哪儿能获得免费的UNIX？

我已经在本文中提到过哪里有最常用的免费UNIX。要想成为一名黑客，你需要兴趣、主动性和自学能力。现在就开始吧……

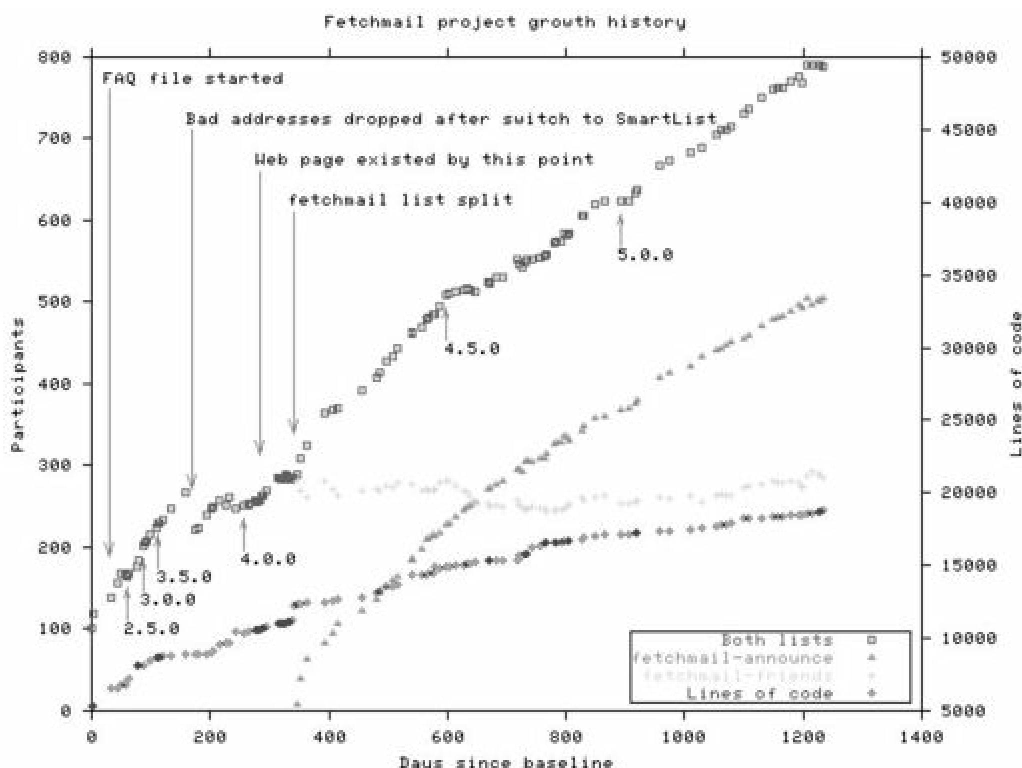
[1] 电脑朋克（cyberpunk），又称赛博朋克、数字朋克，网络叛客等，是cybernetics与punk的结合词。该词最早出现在Bruce Bethke于1983年11月发表在科幻杂志《Amazing》上的短篇小说《Cyberpunk》中。在电脑朋克作家看来，总存在一个统治民众生活的极权主义体系，该体系依靠某种特定的技术（如洗脑、假肢、克隆、遗传工程等方式）来实现统治，这种技术使人和机器结合起来，人们生活的每一个细节都受电脑

网络控制，而生活在社会边缘的局外人（如罪犯、流浪汉、梦想家或只是单纯寻求兴趣爱好的人）与极权主义体系展开斗争。随着时间的演变，电脑朋客通常指那些着迷于电脑的反文化分子，他们喜欢在网络上制造各种恶作剧，借此展示对流行秩序的反抗。——译者注

附录B

fetchmail成长的统计趋势

下面这张散点图使用Gnuplot 3.7制作，数据是用两个shell脚本从fetchmail项目的NEWS文件中抓取的，这两个脚本可以在项目网站上找到。



该图展示了fetchmail项目参与人数的增长。横轴表示的是天数，起始时间是1996年10月，即我开始收集数据的时间，当时的版本号是1.9.0。左纵轴表示参与人数，每次新版本发布都会有一个数据点，因此，图中数据点的密集程度表明了版本发布的频率。

图中一开始出现的峰值（在“Bad addresses dropped（无效地址删

除）”注释之前）似乎是个假象，当时我并没有定期清理变得无效的地址。列表上每个月的流动率约为5%（只是我自己的估计，并没有具体的数据）。

正方形散点表示参与者总人数。十字散点表示的是我把列表分开后在fetchmail-friends里的人数。三角形散点则表示了列表分开后在fetchmail-announce中的人数。

菱形散点代表代码行数（右纵轴），它跟踪了项目的大小。这个散点同其他三个是没有比例关系的。

从图中可以看出，有些趋势是很明显的：

· 项目参与人数随时间呈现出持续的线性增长。

· 项目生命周期中的关键事件是1997年10月4.3.0版本的发布，那时我宣布代码停止开发并进入维护模式，然后我将fetchmail列表分开。

· 项目历史上发布最密集的阶段在4.3.0发布前那段时间（中间的缺口是因为我休了两周的假期），然后出现了一个明显的减速。

· 开发者人数在4.3.0版本以后保持相对稳定，平均大约在250人左右。

· 4.3.0以后的人数增长实质上都发生在announce列表上，也就是那些只使用fetchmail而不参与开发的人。

· 代码规模似乎呈亚线性（sublinear）增长，也可能是对数增长。

人数呈线性增长是非常有意思的。我们之前推测它的增长模型可能是几何级数增长或是逻辑斯谛（logistic）增长，因为这个项目是以口口

相传的。

有人指出，人数之所以呈线性增长，很可能是因为项目数量以及合格程序员人数都以相同的趋势增长（大约是指数增长）。（这样分到每个项目的人数只能呈线性增长。——译者注）

还有一些网页也在做类似的事情：

· http://kitenet.net/programs/debhelper/stats/提供了debhelper工具包的增长统计数据。

· http://durak.org:81/sean/pubs/kfc/提供了Linux内核中某些词汇的统计。

正文注释

黑客圈简史

参考文献

1.David E.Lundstrom."Real Programmer."InA Few Good Men From UNIVAC,1987.An anecdotal history.

2.Levy,Steven.Hackers.Garden City,N.Y.:Anchor/Doubleday,1984.

3.Raymond,Eric S.The New Hacker's Dictionary.Cambridge:MIT Press,1996.

大教堂与集市

注释

1.在《Programing Pearls》（编程珠玑）一书中，著名计算机科学家Jon Bentley对Brooks这个观点的评论是："如果你计划抛弃一个，你会抛弃第二个。"基本上他说的是对的，Brooks和Bentley不只是说你对第一次尝试失败要有所准备，而且强调"用正确的办法重新来过"往往比整理一堆乱麻要有效得多。

2.在互联网井喷之前，就有开源集市开发模式的成功例子，而且和UNIX及互联网传统并无关系。1990年至1992年间info-Zip（主要用于DOS系统的压缩工具，http://www.cdrom.com/pub/infozip/）的开发就是一例；另一例则是诞生于1983年的RBBS公告板系统（也是用于DOS的），它发展出足够强大的社区，尽管互联网邮件和文件分享相比本地BBS有着巨大的技术优势，但直到现在（1999年中后期），RBBS还在相当有规律地发布。info-Zip社区在一定程度上依赖互联网邮件，但RBBS的开发者文化实际上是建立在一个规模很大的RBBS在线社区之上，它完全独立于TCP/IP基础设施。

3.透明性和同行评审对控制操作系统开发的复杂性很有价值，这并不是新观点，1965年，在分时操作系统发展的早期，Multics操作系统的两位设计者Corbató和Vyssotsky写道(http://www.multicians.org/fjcc1.html)：

Multics系统将在大规模试用后再发布……这样做出于两点考虑：第一，系统要经得起公众审视和有兴趣读者的批评；第二，在系统日益复杂的时代，现在和未来的系统设计者有义务让操作系统内部尽可能清晰易懂，以便揭示系统的基本问题。

4.关于重复劳动不一定会拖累开源开发，John Hasler给出了一个有趣的解释，我把它称为“Hasler定律”：重复劳动的代价通常和团队成员数成“次二次方”（sub-quadratically）关系，也就是说，重复劳动的成本比计划和管理成本增长得慢得多，后者更需要想办法消除。

实际上，这个说法和Brooks定律并不矛盾。复杂性成本和bug带来的脆弱性和团队成员数成二次方关系大概是对的。但重复劳动的代价比较特殊，它随团队成员数的变化更慢，这并不难解释，因为人们容易认可这样的不争事实：划定不同开发者代码的功能界限可以避免重复劳动；相比之下，人们并不认为这样就能防范整个系统内各种非预期的不良交互（正是它们导致了大多数bug）。

将Linus定律和Hasler定律结合起来可以发现，软件项目的规模（可分为三种）很关键。对于小项目（我认为也就一到三个开发人员）而言，没有什么管理结构比好好找一个主力程序员更有效。对于规模稍微大一些但传统管理成本相对还比较低的项目，集市模式虽然能避免重复劳动、易于跟踪错误、容易看到不易监管的细节，但其好处并不明显。

从Linus定律和Hasler定律可知，当项目规模大到一定程度，传统管理的成本和问题会比重复劳动的成本增长得快得多。集市模式虽然有自

身问题，但不会像传统模式那样存在不能利用“多眼球效应”（many-eyeballs effect）的结构缺陷。（正如我们所看到的，）集市模式在确认bug和易疏漏细节上比传统模式要管用得多。因此，对于大型项目，这些定律的组合效果，使得传统管理模式的净收益趋近于零。

5.Linux将实验版和稳定版分开的做法，还有一个与风险对冲相关但又截然不同的作用，那就是解决要命的最后期限问题（the deadliness of deadlines）。程序员在需求列表不能调整和最后期限不能拖延的双重要求下，会完全顾不上质量，整个工作很可能会变成一团乱麻。感谢哈佛商学院的Marco Iansiti和Alan MacCormack，他们向我证明，放宽这两个限制的任意一个，都会使进度变得可行。

一种办法是保持最后期限不变而让需求列表灵活一些，允许某些到最后期限时仍未完成的需求被舍弃，这基本上就是“稳定版”核心采取的策略。Alan Cox（稳定版核心的维护人）以相当规律的时间间隔将核心发布，但并不保证某个特定bug何时被修复，也不保证实验版中的某个特性何时会搬到稳定版中。

另一个办法是设定好想要的需求列表，并在其完成时发布，这基本上是“实验版”核心的策略。De Marco和Lister引用研究结果，指出这个进度策略即是“好了告诉我”(wake me up when it's done)，这不仅能够保证最高质量，而且就平均而言，与“保守”或“激进”的进度安排相比，它的交付时间更短。

2000年初，我开始怀疑本文早期版本中我严重低估了“好了告诉

我”这一反最后期限(anti-deadline)策略对开源社区生产力和产品质量的重要性。GNOME 1.0在1999年的匆忙发布给我们上了一课：不成熟发布带来的负面影响，会抵消开源通常所带来的质量收益。

现在看来，开源提升产品质量的三架马车中，“好了告诉我”和开发者“自我选择”是两架，另一架是“过程透明"。

6.Brooks关于解决N平方复杂性问题给出的建议是建立“手术团队”型组织，有人将开源项目“核心+光环”式组织看做是“手术团队”的互联网版本，这种类比很有意思，也不是完全不对，但有很大区别：Brooks设想团队首领周围有一群不同角色的专家（如“代码管理员”），事实上这些角色并不真的存在，一个技术较强的开发者，配上些工具（比Brooks那个年代的工具要强多了），就可以替代Brooks说的这些角色；而开源文化十分倚重的UNIX传统——模块化、API和信息隐藏——都不是Brooks处方上的元素。

7.在谈及bug定位难度时，有位读者曾向我指出，bug跟踪路径长短的差异很大，他推测，对于那种多症状bug，循迹难度将随症状数呈“指数分布”（我认为呈“高斯分布”或“泊松分布”更合理一些）。如果有可能通过实验获得该分布形状的数据，那将会很有价值。由于跟踪的难度远不是一个等概率平均分布，建议即便是单人开发者，也效仿一下集市策略，对每个症状设定一个循迹时间，过了这个时间还找不到，就换另一症状去查。坚持不懈也许并不总是对的……

8.关于能否使用集市模式从零开始一个项目，关键在于集市模式能

否支持真正有创新性的工作。有人声称，如果缺少强有力的领导，集市只能克隆和改进已有最高水平的工程创意，但不能将其推向更高的水平。这也许是万圣节文件(http://www.opensource.org/halloween，两篇令微软尴尬的关于开源现象的内部备忘录)中最臭名昭著的说法，其作者将Linux这种UNIX类操作系统的开发比作是“追逐尾灯”，并认为“（当项目达到最高水平时，）管理也要走在最前沿，而且要强大有力”。

这种说法隐含着严重的事实性错误，比如万圣节文件的作者们后来自己也承认：“通常……，相比其他平台，新的研究成果总是最先在Linux上得到实现和使用。”

Linux并不是开源世界里的新事物，从历史上看，开源社区并不是通过“追逐尾灯”和“强有力管理”发明的Emacs或万维网或互联网本身，而且目前开源方面的创新工作已经多得让开发者都不知道该选哪个好。随手举个例子，GNOME项目正把GUI和对象技术推向最高水平，不仅在Linux社区，在计算机行业媒体中它也获得了强烈的关注。其他例子还有很多，随便哪天你访问一下Freshmeat（http://freshmeat.net/），都会很快证实这点。

有种错误的观点认为教堂模式（或集市模式，或任何其他形式的管理结构）在一定程度上能可靠地产出创意，这完全是无稽之谈。一帮人在一起，不会有什么突破性的创见——即便是集市中那些无政府主义者组成的志愿者团体，通常也不能有真正的创见，更不要说公司委员会中那些为生计着想的还活在过去年代的人了。远见源自个人，其周遭社会

机器的最好做法，就是响应这些有突破性的远见——滋养、奖赏并严格测试它们，而不是压制它们。

有人认为这是对发明家特立独行老派做法的浪漫怀旧，其实不然，我并不是断言一群人不能开发和孵化一个突破性创意，事实上，从同行评审过程可知，正是这类团体开发了高质量的产品。我只是想说，每个这样的团体开发，都必然源自于某人头脑中的一个好创意。大教堂、集市和其他社会结构都可以捕捉这个闪光点并使它更完美，但这些闪光点可不是想要就能有的。

因此创新的根本问题（在软件中，或任何其他领域）是：如何不压制创意。但是，更为根本的是，如何先产生出一批有创见的人。

如果有人认为大教堂模式能够做到这点，而集市模式由于进入门槛低和过程流动性强不能做到这点，那就大错特错了。如果我们需要的是一个人和他的创意，那怎样的社会环境会更有利于这个创意的实现？一个能凭借创意迅速吸引成百上千合作者的环境，必然要好过任何这样的环境：这个人在着手实现他的创意前，不得不向领导层推销他的创意，否则就有被炒掉的风险。

事实上，如果查一下有多少软件创新是因为使用大教堂模型导致的，就很快会发现它非常罕见。大公司靠大学研究提供新创意（所以万圣节文件的作者们对Linux能更快吸收这些研究感到不安），或者靠收购小公司来收购创意，这些都不是大教堂文化的原生创新。事实上，用这种方式移植的许多创新，都在万圣节文件作者们大力颂扬的“强有力

管理”下悄然窒息了。

当然，这是个否定性结论，读者应该看到更具肯定性的结论。我建议：

· 选定一个你认为能始终贯彻的对原创性认定的标准。如果你的定义是“看到了我就能判断”，那也没关系。

· 选择任何一个和Linux竞争的闭源操作系统，然后选一个最佳来源，用于统计该操作系统上的开发工作。

· 对此来源和Freshmeat进行为期一个月的观察。每天统计Freshmeat发布的公告中你认为有‘原创性’的工作，并使用同一标准统计其他操作系统上的公告。

· 30天后，分别对两组数字求和。

我写到这里的当天，Freshmeat发布了22条公告，其中有3个可能在某方面达到了最高水平。对Freshmeat而言这天的成绩并不算好，但是，如果任何读者能发现在任何闭源渠道里一个月内有3条类似的创新，我会十分惊讶。

9.现在，我们有了一个比fetchmail能更好的对集市模式进行验证的系统：EGCS，http://egcs.cygnus.com/,即实验性GNU编译系统。

1997年8月中旬，EGCS项目问世了，它是一个对“大教堂与集市”早期版本中的观点进行有意识尝试的项目。项目创始人觉得GCC（GNU的C编译器）开发已经停滞不前。之后20个月内，GCC和EGCS成为并行产品——二者均从互联网上吸纳开发人员，均源于相同的GCC源码

库，均使用了几乎一样的UNIX工具箱和开发环境。二者不同之处仅在于EGCS有意识地尝试运用我之前描述的集市策略；而GCC则保留一种更类似大教堂模式的组织结构，其开发团队封闭，而且不常发布新版本。

这很像是一个应要求而开展的对照实验，其结果很有戏剧性，数月之内，EGCS版本在一些特性上已经有了实质性领先，比如更好的优化，以及对FORTRAN和C++的更好支持。很多人发现EGCS的开发快照要比GCC的最新稳定版本更可靠，而主要的Linux发行版已经开始转向EGCS。

1999年4月，自由软件基金会（GCC的官方赞助商）解散了原有GCC开发小组，并正式将该项目控制权移交给EGCS指导小组。

10.当然，Kropotkin批判和Linus定律引发了人们对社会组织控制论的更广范思考。比如软件工程的一个通俗理论（Conway定律）常常被表述为“如果有四个小组致力于开发一个编译器，那么你将会得到一个四步（4-pass）编译器。”其原始陈述更具一般性：“如果一个系统由多个组织共同设计，那么其设计的系统将不得不成为这些组织间沟通结构的拷贝。”我们可以更简洁地称之为“方法决定结果”，或者乃至“过程变成产品”。

值得注意的是，开源社区的组织形态和功能在很多层面上是匹配的。网络就是一切，而且无处不在；不只是互联网，还包括在其上工作的人，这些人形成了一个分布式、松耦合以及点到点的网络，它很优雅

地实现了多重冗余和去等级化。这两个网络中，一个节点只有在其他节点想要和它合作时才显得重要。

点到点结构是开源社区出现惊人生产力的关键。Kropotkin曾经对权力关系做出的评价被“SNAFU原则”[1]进一步阐明：“只有平等个体之间才有真正的交流，因为下级向上级讲好听的谎话会比讲真话更能得到持续的奖励。”真正的交流是创造型团队必不可少的，而权力关系将极大制约这点。开源社区有效避免了这种权力关系，并通过对比告诉我们权力关系会带来多么糟糕的代价：大量的bug、低下的生产力和机会的丧失。

更进一步，“SNAFU原则”预测在权力组织中，决策者和现实情况会越来越脱节，因为决策者会听到越来越多好听的谎言。在传统软件开发中很容易看到这一点，由于下级有强烈的动机去隐藏、忽略和最小化问题，当这个“过程变成产品”时，软件会成为一个灾难。

参考文献

我引用了Frederick P.Brooks经典之作《人月神话》中的一些内容，在很多方面，他的观点还需要改进。真心推荐由Addison-Wesley出版社发行的25周年纪念版，这版中新增了Brooks在1986年发表的“No Silver Bullet”。

新版中还有一篇非常有价值的Brooks的回顾，他在20年后对该书重新思考，并坦率承认原文中有几个观点没有经受住时间的检验。我首次读到这篇回顾文章时（在本文首个公开版本大致完成时），惊讶地发现

Brooks把集市类（bazaar-like）实践归功于微软！（事实上，这种归功后来被证实是错误的。1998年，我们从万圣节文件（http://www.opensource.org/halloween/）中了解到微软内部的开发社区是被严重分割的，他们甚至都没有集市模式所需要的共享源码库。）

Gerald M.Weinberg在《The Psychology of Computer Programming》（New York:Van Nostrand Reinhold,1971）一书中介绍了“无私编程”（egoless programming）的概念，虽然这个命名有点令人遗憾。早在他之前就有人认识到“命令原理”（principle of command）的无用性，但Weinberg可能是在软件开发领域最早意识到并提出这点的人。

Richard P.Gabriel对UNIX文化进行了深入的思考（早在Linux时代之前），1989年他在“LISP：好消息，坏消息，如何成为大赢家”一文提出了一个简单的集市类模型，并不太情愿地论证了它的优越性。虽然这篇文章略显过时，但它仍然在LISP粉丝中（包括我）受到追捧。一个读者提醒我说“Worse Is Better”那节几乎是对Linux的预言。这篇文章可以在http://www.naggum.no/worse-is-better.html中找到。

De Marco和Lister的著作《人件》（New York:Dorset House,1987）是一篇被低估了的佳作，我很高兴看到Fred Brooks在他的回顾文章中引用了该书论点。尽管该书作者极力主张的观点并不能直接应用于Linux或开源社区，但其对创造性工作必要条件的洞见是敏锐的，对任何想要把集市模型优点融入商业环境的人来说，该书都值得一读。

最后，我必须承认我差点把本文命名为“The Cathedral and the

Agora”，Agora在希腊语中是公开市场或公共聚会场所的意思。Mark Miller和Eric Drexler在多篇论文中提出了有开创性意义的“agoric systems”，描述了这种市场类型计算生态的突出特点，这使得在5年后Linux引起我兴趣时能够让我对开源文化中类似现象进行清晰的思考。这些论文可以在http://www.agorics.com/agorpapers.html中找到。

致谢

本文的改善要归功于和很多人的交流以及他们的排错帮助。特别感谢Jeff Dutky（dutky@wam.umd.edu）提出的关于“排错可以并行”的构想，并帮助沿此构想进一步分析。同样感谢Nancy Lebovitz（nancyl@universe.digex.net），是她建议我效仿Weinberg引用Kropotkin的文字。感谢General Technics列表中Joan Eslinger（wombat@kilimanjaro.engr.sgi.com）和Marty Franz（marty@net-link.net）提出的敏锐批评。Glen Vandenburg（glv@vanderburg.org）指出了贡献者自我选择的重要性，并提出多流程开发可以修复“疏漏型bug”这个颇具意义的观点。Daniel Upper（upper@peak.org）提出了关于这点的自然类比。非常感谢PLUG（费城Linux用户组）成员，他们提供了本文首发版的读者测试。Paula Matuszek（matusp00@mh.us.sbphrd.com）在软件管理实践方面给予我指导。Phil Hudson（phil.hudson@iname.com）提醒我：黑客文化的社会组织反映了其软件的组织结构，反之亦然。John Buck（johnbuck@sea.ece.umassd.edu）指出了MATLAB与Emacs之间有

启发性的类比。Russell Johnston（russjj@mail.com）使我意识到在“多少眼睛驯服复杂性”中讨论的一些机制。最后，Linus Torvalds的意见很有帮助，他很早就对本文表示认可，这实在令人鼓舞。

开垦心智层

注释

1.“心智层”（noosphere）是一个比较模糊的哲学术语。它的发音是KNOW-uh-sfeer（两个o分别发音，一个是长的重音，一个是短的非重音（接近于非中央元音）。如果对表音法非常在意的话，可以第二个“o”上面适当地标一个分音符，以表明它是一个单独的元音。

详细说来，该术语是指“人类思想的总和”，源于希腊语“noos”，其含义是“理智”、“智慧”或“气息”。它由E.LeRoy在《人类起源和智力的进化》（Paris,1928）中创造，最初由俄罗斯生物学家及杰出生态学家Vladimir Ivanovich Vernadsky（1863-1945）推广，并由耶稣会信徒古生物学家/哲学家Pierre Teilhard de Chardin（1881-1955）进一步普及。它现在常和Teilhard de Chardin关于未来演化的理论联系在一起，该理论认为人类进化将达到一种和神性合一的极致的纯净心智。

2．David Friedman是当代经济学研究中论述最清晰易懂的思想家之一，他撰写了一篇极好的关于知识产权法历史与逻辑的概论（http://www.best.com/~ddfr/Academic/Course_Pages/L_and_E_LS_98/Why
我建议所有对这些话题感兴趣的人都可以把它作为入门读物。

3.将Linux与BSD相比，一个有意思的不同之处是：Linux内核（以及与OS内核相关的实用工具）从来没有分裂过，但BSD内核至少分裂过三次。之所以有意思，是因为BSD社区的社会结构是集中式的，其意图

在于定义明确的权力界限并防止分裂，但去中心化和无组织的Linux社区并没有这些措施。这么看来，似乎开放式开发的项目实际上最不可能产生分裂！

Henry Spencer（henry@spsystems.net）指出：通常，政治体系的稳定性与其政治过程的准入门槛高度是成反比的。他的分析值得在这里引用：

对于相对开放的民主制度而言，它的一个主要优势在于，绝大多数潜在的革命者发现通过在系统中工作比攻击该系统更容易让自己向目标前进。但如果既有政党联合起来“提高门槛”，导致那些较小的不满意团体觉得更难实现自己目标的话，这种优势就很容易被侵蚀破坏。

（在经济学中可以找到一个类似的原理。开放市场有着最强的竞争力，并且通常有着最好和最便宜的产品。因此，既有公司有很强的利益动机提高市场准入难度，例如游说政府对电脑进行详尽的RFI测试，或者建立非常复杂以至于没有强大资源就无法从头开始实施的“共识”标准。拥有最严格准入门槛的市场正是那些遭受革命者最猛烈攻击的市场，例如互联网和美国司法部与贝尔系统之间的对战。）

准入门槛不高的开放过程鼓励参与而非分裂，因为参与者能从中获得成果，而不用付出分裂所需的高昂成本。尽管这种成果可能不像分裂所得成果那样令人印象深刻，但其成本较低，且大多数人都能接受这种折衷。（当西班牙政府取消佛朗哥的反巴斯克法令并允许巴斯克多省可以拥有自己的学校和一定的自主权时，绝大多数巴斯克独立运动几乎一

夜之间消失。只有极端的人坚持认为这还不够好。）

4.流氓补丁是一个比较微妙的话题。可以将其分为“友好型”和“非友好型”两种。“友好型”补丁在设计时会考虑合并回项目主线并受项目维护者控制（不论这种合并是否会真的发生）；“非友好型”补丁则试图把项目拉往维护者不同意的方向。一些项目（尤其是Linux内核自身）对“友好型”补丁的态度非常宽松，甚至鼓励它们在其自身beta测试时独立发布。而“非友好型”补丁则表明了是要和原有版本竞争，这是很严肃的问题，如果有人或组织维护了成规模的“非友好型”补丁，则往往意味着要走向分裂。

5.非常感谢Michael Funk（mwfunk@uncc.campus.mci.net）指出将黑客文化和破解（pirate）文化进行比较会很有启发意义。Linus Walleij在他发表的“A Comment on'Warez D00dz'Culture”（http://www.df.lth.se/~triad/papers/Raymond_D00dz.html）一文中，对破解文化的动力做了分析，提出了和我不同的观点（他认为该文化是稀缺经济文化）。

这种对比可能不会持久。前骇客Andrej Brandt（andy@pil-grim.cs.net.pl）认为骇客/warez d00dz文化正在枯萎，因为其中最聪明的成员和领导者正在被开源世界同化和吸收。骇客团体“Cult of the Dead Cow”在1999年7月那次史无前例的行动，也许为此观点提供了独立的证据，他们发布了突破微软安全机制的“Back Orifice 2000”及其源码，并遵循GPL。

6.从进化学角度讲，工匠所具备的努力奋进的性格特点（就像内化的道德）源自于欺骗的高成本和高风险。进化心理学家搜集的实验证据表明，人类拥有专门识别社会欺骗行为的大脑逻辑，我们也很容易明白为什么我们的祖先会因能够识别欺骗而被进化选择。因此，如果一个人希望因为好的人格特质而获得声誉并进而取得好处，但又想避免风险或代价，那么最好的策略也许就是实际拥有而非假装拥有这些特质。（“诚实是最好的策略。”）

进化心理学家的这个观点解释了酒吧斗殴之类的行为。对年轻的成年男性而言，如果拥有身体强壮的声誉，无论是在社交上还是在与异性交往上（即便在如今女权主义影响的大气候下）都会有用。然而，假装身体强壮则非常冒险，如果被发现，会让伪装者处于更加糟糕（与从未声称拥有该特质相比）的地位。欺骗的代价如此之高，所以有时更好的极大极小策略[2]是：将强壮这种特质内化并冒着在打斗中受重伤的风险来证明它。对“诚实”这类争议较少的特征，也有类似的观测结论。

创造性工作会给创作者带来类似冥想那样的奖赏，这点我们不应低估，但工匠的努力奋进至少有一部分是这种内化（其基本特质为“极其认真勤奋”或类似的能力）的表现。

不利条件理论（handicap theory）可能也与此有关。雄孔雀华而不实的尾巴和雄鹿巨大的鹿角在雌性看来性感迷人，因为它们表达出雄性健康（因而适合产出健康的后代）的信息。就像是在说：“我精力很充沛，这种展示虽然很奢侈，也很浪费能量，但我负担得起。”赠送源

码，就像拥有一辆跑车，非常类似这种奢华的展示——昂贵但没有明显的回报，但这让当事者至少在理论上非常性感迷人。

7.可以在http://www.valdosta.peachnet.edu/~whuitt/psy702/regsys/maslow.html上查阅和马斯洛需求层次相关理论的简明摘要。

8.然而，要求领导谦逊也许是礼物文化或富裕文化的一个普遍特性。David Christie（dc@netscape.com）记录了他在穿越斐济外岛的旅行中的见闻：

“在斐济土著村落的酋长身上，我们看到了与开源项目领导人类似的自谦和低调的领导风格。尽管酋长备受尊敬并拥有所有实际权力，但我们遇到的酋长都表现出真正的谦逊和一种圣徒般的敬业态度。尤为让人感兴趣的是，酋长是世袭的，不是被选举的，也不是人气竞赛（popularity contest）产生的，他们生下来就是酋长而无需竞争，但不管怎样，文化教导他们做到了这点。”他还强调他相信斐济酋长的这一特点源自于强迫他人合作的困难性：酋长没有胡萝卜，也没有大棒。

9.一个不可忽略的事实是：成功创建项目的人，比起那些可能做了相同工作量的调试者和协助者，拥有更高的威信。本文早期版本提出一个问题，即“这种对比是否合理？或者这是人们潜意识中的领土模型的二阶效应？”一些读者提出了有说服力的但本质上相同的理论，如Ryan Waldron（rew@erebor.com）在下面的分析中就给出了很好的阐述：

在Lockean土地理论的语境下，某人如果成功创建了一个新项目，他

（她）在实质上就是发现或开辟了一片可以让他人开垦的新领土。对大多说成功项目而言，都存在回报衰落模式，所以在一段时间后，通过对项目贡献而获得的荣誉会越来越稀少，后来者很难获得较高的声誉，而不论其工作有多么出色。

比如，如果我对Perl代码做改进，那我得做得多好，才能获得别人对我的认可？哪怕是得到Larry、Tom、Randall他们已获得的很小一部分？

然而，如果有人明天创建一个新项目，我能早早加入并经常参与，那么因为参加得早，我分享该项目成就的能力就会极大增强（假设贡献的质量相同）。我认为这与早些或晚些投资微软股票很类似。大家都能赚钱，但较早的参与者获利更多。因此，在某种程度上，比起投资一个持续增长的公司股票，我对一个新的并且成功的IPO更感兴趣。

这种类比还可以继续展开，项目创始人通常都得宣传和推销他的创意，这个创意可能会也可能不会被他人认可。因此，项目创始人遇到类似IPO的风险（有可能损害其声誉），其风险要大于协助一个已经在同侪中获得认可的项目。创始人能通过项目获得持续不断的回报，虽然事实上那些协助者可能做了更多的实质性工作。很明显，这类似于交换经济中风险和回报的关系。

还有些读者提到，我们的神经系统更善于察觉差别而非稳态。所以，新项目带来的创新性变革，会比持续改进带来的累积效应，更容易被世人发现。因此，Linus被人们尊为Linux之父，虽然数以千计贡献者所做改进的总体效果，对OS成功的贡献，远远超过一人所能做到的。

10.“专有化”（de-commoditizing）一词来自万圣节文件（http://www.opensource.org/halloween/），微软在其中不加掩饰地使用该词，点明了他们最为有效的长期策略，并以此维持他们对用户的垄断锁定和利润榨取。

11.一位读者指出“只有其他黑客认为你是黑客时，你才是黑客”所表达出来的价值观，类似于中世纪欧洲骑士时代精英团体所宣称的理想（通常总是无法实现），这些社会精英有足够的财力超脱于所处的稀缺经济。像人们期望的那样，一个有抱负的骑士为正义而战，他追求荣誉而非获取钱财，他站在弱势和受压迫者一边，不断寻求机会挑战自己英勇才能的极限。也正因如此，他会认为自己（并被别人认为）是人中翘楚——前提是他的能力和品行已获得其他骑士的承认和正式认可。在亚瑟王传奇（Arthurian tales）和武功歌（chansons de geste）所颂扬的骑士精神中，我们看到了理想主义、对自我的不断挑战和对地位的追求，这与当今激励黑客的要素是类似的。看来，类似的价值观和行为规范总是围绕这样的技能而演化：它需要极大的奉献精神，同时又充满能量。

12.自由软件基金会主站（http://www.gnu.org/philosophy/motivation.html）上面有一篇文章，总结了许多此类研究的结果。本文的部分引用源自于此。

参考文献

Miller,William Ian.Bloodtaking and Peacemaking:Feud,Law,and Society in Saga Iceland(Chicago:University of Chicago Press,1990)。这是一项关于

冰岛群众集会（folkmoot）法律的引人入胜的研究，它阐释了洛克所有权理论的渊源，并描述了从习惯到习惯法再到成文法这一历史进程的后一阶段。

Malaclypse the Younger.Principia Discordia,or How I Found Goddess and What I Did To Her When I Found Her(Loompanics,1980)。文中描述的Discordianism教义里，有很多看似无厘头但却很具启发意义的观点，其中“SNAFU原则”相当犀利地分析了命令体系不能很好扩展的原因。这里是一个可浏览的HTML版本，http://www.cs.cmu.edu/~tilt/principia/。

Barkow,J.L.Cosmides,and J.Tooby(Eds.).The Adapted Mind:Evolutionary Psychology and the Generation of Culture(New York:Oxford University Press,1992)。这本书很精彩地介绍了进化心理学。其中有些文章和我所讨论的三种文化类型（命令/交换/礼物）直接相关，这些模式也许已经植入了人类的心灵深处。

Goldhaber,Michael K.:“The Attention Economy and the Net”,（http://www.firstmonday.dk/issues/issue2_4/goldhaber)。完成本文1.7版本之后我才发现这篇文章，它存在一些明显的缺陷（Goldhaber认为“用经济学分析注意力是不适用的”，这一观点经不起仔细推敲），但仍然很有趣并充满洞察力地阐释了“寻求注意”在组织行为中的作用。我所讨论的威望或同侪声誉可以看作是他所认为的注意力的一个特殊情况。

我已经在“黑客圈简史”（http://www.tuxedo.org/~esr/faqs/hacker-hist.html）中总结了黑客文化的历史。需要有人写本书把它好好阐释一

下，但应该不会是我。

致谢

Robert Lanphier（robla@real.com）对“无私行为”讨论的贡献颇多。Eric Kidd（eric.kidd@pobox.com）强调，提倡谦逊的价值观有利于防范个人崇拜。全球性影响这一部分受到Daniel Burn（daniel@tsathog-gua.lab.usyd.edu.au）评论的启发。文化移入这部分的思路受到Mike Whitaker（mrw@entropic.co.uk）的启发。Chris Phoenix（cphoenix@best.com）指出了“黑客无法通过贬低其他黑客而赢得声誉”这一事实的重要性。A.J.Venter（JAVen-ter@africon.co.za）给出了与中世纪骑士精神的类比。Ian Lance Taylor（ian@airs.com）发来了对声誉竞争模型的详细批判，这促使我透彻思考并更清晰地说明这个假定。

魔法锅

注释

1.如果假设编程人才均匀分布在项目用户（随时间不断增长）之中，那么供应不足问题事实上就与用户数之间成线性关系，但事实上这并不是问题所在。

在注释2中探讨的动机（以及一些更传统的经济动机）暗示有才能的人倾向于寻找那些与他们兴趣相匹配的项目，而项目也在寻找他们。因此，理论上讲（经验也倾向于证实这点），最有价值（最有资质和最有积极性）的人才在寻找适合他们的项目时，通常会选择在项目生命周期的相对初期阶段，越往后越少。

虽然缺乏具体的数据支持，但基于以往经验，我强烈倾向于认为：一个项目在生命周期成长过程中吸收人才的规律符合典型的逻辑斯谛曲线。

2.Shawn Hargreaves在“Playing the Open Source Game”（http://www.talula.demon.co.uk/games.html）一文中就游戏软件是否适合开源给出了很好的分析。

3.给会计人员：如果不用定值美元（constant dollar）而用贴现后现值（discounted present value）计算，“服务成本终将超过固定预付价格（fixed up-front price）”这一论断仍然有效，因为未来销售收入和服务成本有着同样的贴现率。

一个类似但更微妙的相反论点是，对于每个软件拷贝，当买家停止使用时，服务成本将会变为零；因此如果用户在他（她）还没有产生太多服务成本之前就停止使用软件的话，你就还是赚的。基本上该论点仍在表达“工厂模式鼓励存架软件”。或许更有指导意义的说法是：服务成本超过销售收入的风险随着软件预期使用时间的增长而变大。这也就是说：工厂模式对质量是一种惩罚。

Wayne Gramlich（Wayne@Gramlich.Net）指出，工厂模式之所以长期存在，一定程度上是因为会计准则的落后和陈旧，会计准则形成的年代里，机器和建筑物比人力更重要。在软件公司的账目上，一般将电脑、办公设施以及建筑等列为公司资产，而程序员则属于开支。而事实上，程序员才是真正的资产，电脑、办公设备和建筑物则无关紧要。这种扭曲的估值方法来自美国国税局和股票市场的压力，其目的是维持稳定、统一的会计准则，以减少用货币给公司估值时的复杂性，但其结果导致会计准则跟不上现实情况。

基于这一观点，给产品标上高价（而不考虑未来的服务价值）在一定程度上是一种防卫机制，而所有相关方也一致假装认为标准的会计准则并没有在本体论层面上站不住脚。

（Gramlich还指出这些准则使得一些软件公司在IPO之后很快出现奇怪的而且通常是自毁式的大肆收购：“通常软件公司发行一些额外的股票来筹措竞争基金。但是他们不会从中拿出任何一点来扩充编程队伍，因为会计准则会认为这是开支的增加。所以，新上市的软件公司不得不

通过收购其他软件公司来发展壮大，因为收购行为在会计准则看来是一种投资。”）

4.OpenSSH是开源项目变节后产生分支的典型例子，查看一下它的历史可知，在SSH(Secure Shell)走向封闭之后，OpenSSH从SSH的先前版本中分支而出。

参考文献

The Cathedral and the Bazaar,http://www.tuxedo.org/~esr/writings/cathedral-bazaar/

Homesteading the Noosphere,http://www.tuxedo.org/~esr/writings/homesteading/

De Marco and Lister.Peopleware:Productive Projects and Teams.New York:Dorset House,1987.

致谢

感谢David D.Friedman，与他几次极具启发性的讨论使我完善了开源合作的“反公地模型”。感谢Marshall van Alstyne，他指出竞争性信息在概念上的重要性。感谢Indiana Group的Ray Ontko提出的有益批评。感谢1999年6月之前一年里聆听我演讲的很多听众，他们对本文也有帮助，如果你是他们中的一员，你知道我说的是什么。

本文首次发布后几天内收到的电子邮件回馈，帮助了本文的实质性改善，这可以看做是开源模式的又一个例证。Lloyd Wood指出开源软件的重要性在于它能“防患于未然”；Doug Dante提醒我关于“未来免费”的

商业模式；Adam Moorhouse提出的问题引发我们对闭源回报的讨论；Lionel Oliviera Gresse帮我对其中一个商业模式想出了好名字；Stephen Turnbull对我未能严谨论述“搭便车”效应而产生的愚蠢错误给予了无情批评；Anthony Bailey和Warren Young帮我纠正了Doom案例中的一些错误；Eric W Sink敏锐地指出工厂模式在事实上鼓励了存架软件。

延伸读物

有关开源的学术性分析已逐步出现，一些相关材料可以在我的网页上找到：http://www.tuxedo.org/~esr/writings/cathedral-bazaar。

Ross Anderson.How to Cheat at the Lottery（或Massively Parallel Requirements Engineering）。在这篇富有洞察力、逻辑清晰而又生动有趣的文章中，作者给出了应用集市模式开展并行工作的一个实验性结果，它并非用于编码，而是用在计算机安全领域中一个难题的需求分析和系统设计上。参见：http://www.cl.cam.ac.uk/~rja14/lottery/lottery.html.

Davis Baird."Scientific Instrument Making,Epistemology,and the Conflict between Gift and Commodity Economies."InJournal of the Society for Philosophy&Technology,Volume 2,no.3-4.这篇文章很有趣，它完全没有谈及软件或者开源，其研究建立在较早的关于礼物文化的人类学文献上，它给出了在很多方面与“开垦心智层”中相类似的分析。参见：http://scholar.lib.vt.edu/ejournals/SPT/v2_n3n4html/baird.html

Asif Khalak,Evolutionary Model for Open Source Software:Economic Impact.作者尝试分析性地对开源市场渗透进行建模，并用计算机仿真来测试此模型对各种成本和行为参数的依赖。这篇文章在1997年7月“遗传与进化计算大会”（Genetic and Evolutionary Computation Conference）的博士研讨会上被介绍。参见：http://web.mit.edu/asif/www/ace.html

Bojidar Mantarow.Open Source Software as a New Business Model.作者

以红帽软件为案例，阐述了降低成熟市场准入壁垒的效果。这篇论文提交于1999年8月，是作者在雷丁大学（University of Reading）提交的国际管理专业理学硕士论文的一部分。参见：
http://www.lochnet.net/bozweb/academic/dissert.htm

Eben Moglen.“Anarchism Triumphant:Free Software and the Death of Copyright.”这篇文章最初发表在“哥伦比亚法律评论”（Columbia Law Review）上，很遗憾其中存在大量事实上和逻辑上的错误，而且其分析内容几乎被其误导的政治辩论所扼杀。但不管怎样，这是一篇有趣而刺激的读物，即使只是为了Moglen那令人难忘的对法拉第定律的推论也值得通篇阅读：“将互联网围绕在每个人的大脑上，并让这个星球旋转，软件就会在网线间流动。”参见：
http://old.law.columbia.edu/my_pubs/anarchism.html

[1] SNAFU是Situation Normal All Fucked Up的缩写，指事态已经混乱不堪，但从管理上看一切正常。该用语源于军队，因为士兵向长官汇报时经常说“Situation Normal”。——译者注

[2] 极大极小（minimaxing）策略是指博弈论中让参与者的风险极大值达到极小的策略。——译者注